DESIGN AND IMPLEMENTATION OF A MODIFIED MEDIAN ROUND ROBIN ALGORITHM (DIMMRRA)

BY

HAFSATU MORA

DEPARTMENT OF MATHEMATICS
AHMADU BELLO UNIVERSITY,
ZARIA, NIGERIA.

APRIL, 2016

DESIGN AND IMPLEMENTATION OF A MODIFIED MEDIAN ROUND ROBIN ALGORITHM (DIMMRRA)

BY

HAFSATU MORA

B.Sc. (UDUS, 2000)

M.Sc/SCI/8661/2011-12

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES,

AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF

MASTER OF SCIENCE (MSC) DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS,

FACULTY OF SCIENCE,

AHMADU BELLO UNIVERSITY,

ZARIA. NIGERIA

APRIL, 2016

Declaration

I hereby declare that the work in this Dissertation titled ”Design and Implementation of a Modified Median Round Robin Algorithm (DIMMRRA)” was performed by me in the Department of Mathematics, under the supervision of Dr. S. E. Abdullahi and Prof. S. B. Junaidu. The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this work has been presented for another degree or diploma at any institution.

HAFSATU MORA	_____	_____
Name of Student	Signature	Date

Certification

This Dissertation titled "Design and Implementation of a Modified Median Round Robin Algorithm (DIMMRRA)" meets the regulations governing the award of the Degree of Master of Science of the Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Dr. S. E. Abdullahi	_____	_____
Chairman, Supervisory Committee	(Signature)	Date
Professor S. B. Junaidu	_____	_____
Member, Supervisory Committee	(Signature)	Date
……	_____	_____
External Examiner	(Signature)	Date
Professor A. A. Tijjani	_____	_____
Head of Department	(Signature)	Date
Professor Kabiru Bala	_____	_____
Dean, School of Postgraduate	(Signature)	Date

Dedication

I dedicated this research work to my beloved husband Abubakar Ahmadu Kurfi, for his moral support and spiritual encouragement rendered to me.

It is also dedicated to my beloved late father Alhaji (Dr.) Abdurrahman Mora of blessed memory. May Almighty Allah in his infinite mercy grant his gentle soul eternal rest, Amin.

Acknowledgement

Glory is to Allah (SWT) the Lord of the Worlds, for giving me the gift of life and health to make this work a success. Alhamdulillah.

My wholehearted appreciation and immense gratitude goes to my Major supervisor Dr. S. E. Abdullahi who showed me the interest of choosing Operating System as an area of research and for diligently going through this work to make it a success and worthwhile.

I wish to express my profound gratitude to my Minor supervisor Professor S. B. Junaidu whose door is always open for his useful suggestions and encouragement while writing this research work.

I remain indebted to my loving husband Abubakar Ahmadu Kurfi for his encouragement, patience and understanding, without his support, this research would have been difficult if not impossible. May the Almighty Allah reward you abundantly in this world and in the hereafter, Amin.

I am duty bound to thank my parents, late Alhaji Dr. Abdurrahman Mora and my mother Hajiya Binta Abubakar Imam for their parental, moral support since I was born. I could never forget to mention what my father used to say "***there is nothing a parent could give to his child rather than education***" and Alhamdulillah he gave us that. May his soul rest in perfect peace, and may the Almighty Allah give long live and prosperity to my caring mother, Amin.

My unreserved and golden appreciation goes to my children: Sumayya A. Kurfi; Abdurrahim A. Kurfi; Ummu Sa'ada A. Kurfi; Abdulhaleem A. Kurfi and little Fatuma A. Kurfi for their support and prayers. They really stood to see that my coming to Zaria must be a success. Thank you and May Allah reward you.

Also, I deem it very much necessary to use this Avenue to show my appreciation to the entire Mora family for their encouragement.

Finally, my sincere appreciation goes to my friend Amina Hassan Gusau and her husband Dr. AbubakarYahaya. I cannot find words good enough to thank you all for your support. I pray to Almighty Allah to help and guide us all to achieve the best in all our endeavors, Amin.

Wassalamu Alaikum.

Hafsat Mora, April, 2016.

Abstract

Central Processing Unit (CPU) scheduling involves a careful examination of pending processes to determine the most efficient way to service the requests. Several scheduling algorithms have been designed to arrange accesses to computer resources efficiently. Round robin scheduling algorithm (RRSA) is an attractive algorithm but suffers from the problem of time quantum determination. In the classical round robin algorithm, quantum time is fixed throughout the scheduling process. This static nature made it very difficult to optimize the algorithm. The only solution to this problem is to provide a quantum time that changes dynamically during execution. Major challenges of dynamic round robin schedulers reported in the literature are: they do not include Average Response Time as a criterion for comparison and they do not bother much on preempting processes with negligible completion time after executing for a given time quantum leading to an increase in the number of context switches. This research proposed an algorithm, the Modified Median Round Robin (MMRRA), with a dynamic time quantum aimed at reducing the overhead of the RRSA. The proposed algorithm was implemented and evaluated against the following five algorithms in the literature: Improved Round Robin (IRR), Improved Mean Round Robin with Shortest Job First (IMRRSJF), Dynamic Round Robin with Controlled Preemption (DRRCP), Half Life Variable Quantum Time RR (HLVQTRR) and CLASSICAL RR. Which in turns, proved to perform better in terms of AWT, ATAT and NCS. But, in terms of ART, HLVQTRR has the best result but still the result for MMRRA was not bad.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AMMRR: Average Mid Max Round Robin

AMRR: Average Max Round Robin

ART: Average Response Time

ATAT: Average Turnaround Time

AVG: Average

AWT: Average Waiting Time

BT: Burst Time

CPU: Central Processing Unit

CS: Context Switch

DRRCP: Dynamic Round Robin with Controlled Preemption

FCFS: First Come First Serve

FIFO: First In First Out

FJFDRR: Fittest Job First Dynamic Round Robin

HB: Highest Burst

HLVQTRR: Half Life Variable Quantum Time Round Robin

i: Loop Variable

IMRRSJF: Improved Mean Round Robin with Shortest Job First

I/O: Input/output

IRR: Improved Round Robin

MD: Median

METHOD: Way of solving a problem.

MMRRA: Modified Median Round Robin Algorithm

N: Number of Process

NCS: Number of Context Switching

PBT: Process Burst Time

PCB: Process Control Block

Pi: i^{th} Process:

QT: Quantum Time

RAM: Random Access Memory

RBP: Remaining Burst Time of Process

RR: Round Robin

RT: Response Time

SIMMRRA: Study and Implementation of a Modified Median Round Robin Algorithm

SJF: Shortest Job First

SJN: Shortest Job Next

SRT: Shortest Remaining Time

T: Turn-Around Time

VB: Visual Basic

WT: Waiting Time

Z: Random Variable

μ: Mean

$\acute{o}$: Standard Deviation

CHAPTER ONE: INTRODUCTION

1.1 BACKGROUND OF THE STUDY

Process scheduling algorithm has been an interesting field of study in Operating Systems. Scheduling is a key concept in computer multitasking, multiprocessing and real-time operating system designs.

The operating system uses some sort of scheduling techniques to allocate resources to processes in the ready queue. Scheduling refers to the way processes are assigned to run on available Central Processing Unit (CPU). Certain CPU scheduling techniques or algorithms have been developed. The overall goal of these algorithms is to: maximize CPU utilization, reduce average waiting time and average turnaround time, reduce the number of context switching, increase throughput and try as much as possible to be fair to all processes.

Processor Manager is in charge of allocating a single CPU to execute the jobs of those users.In a single user system, the processor is busy only when the user is executing a job but, at any other time it is idle. Processor Management in this environment is simple. However, when there are many users with many jobs on the system (this is known as a multiprogramming environment), the processor must be allocated to each job in a fair and efficient manner, which can be a complex task. The processor, also known as the CPU (Central Processing Unit), is the part of the machine that performs the calculations and executes the programs.

Multiprogramming requires that the processor be allocated to each job or to each process for a period of time and de-allocated at an appropriate moment. If the processor is de-allocated during a program's execution, it must be done in such a way that it can be restarted later as easily as possible, this is a delicate procedure.

A single processor can be shared by several jobs or several processes, but if, and only if, the operating system has a scheduling policy, as well as a scheduling algorithm, to determine when to stop working on one job and proceed to another. Some examples of these scheduling algorithms are:

i First Come First Serve **(FCFS)**: Attends to processes in their order of arrivals.

ii Shortest Job First **(SJF)**: Attends to processes in an increasing order of their burst time.

iii Round Robin **(RR)**: Gives an equal time slice (Quantum) to every process in the ready queue in a circular manner. If 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 time Quantum and then return to the tail of the ready queue to take turn.

iv Priorities scheduling: Here, processes are attended to based on their priorities.

The preemptive nature of Round Robin 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 too much context switching is an overhead. 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. Therefore, an optimal Quantum time is what we need.

In the ***Classical RR*** the quantum time is static. This is why it is sometimes called Static RR. Here, the quantum time is gotten from the average of the processes' burst time in the ready queue and remain constant throughout the entire procedure. The static nature made it difficult for maximum improvement, and as such research on ***Dynamic RR*** is on. The idea of dynamic RR is to use more than one different quantum time for processes depending on their circumstances. Some of the proposed dynamic RR are: Improved Mean Round robin with shortest job first scheduling, Self-Adjustment time quantum in Round Robin algorithm depending on Burst time of the New Running Processes (SARR), An Improved Round Robin CPU Scheduling Algorithm (IRR), A new Round Robin Based Scheduling Algorithm for Operating System, Dynamic Quantum Using the Mean (An Algorithm), Half-life variable quantum time RR (HLVQTRR), Dynamic Round Robin with Control Preemption (DRRCP) etc. So far, from the various proposed Dynamic RR, it has been proven to be more efficient than the Classical RR when compared and evaluated on the basis of number of context switching, average waiting time and average turnaround time.

The purpose of this research is to develop a new dynamic RR algorithm base on the ideas gotten from the different scheduling algorithms like IRR, IMRR, HLVQTRR, DRRCP etc. that will provide answers to some of the critical setbacks experienced in the classical RR and the various proposed dynamic RRs. Comparative analysis would be carried out on Six RR algorithms to enable this research work reach a meaningful conclusion.

1.2 PROBLEM STATEMENT

The greatest challenge in RR is Quantum time. That is, what should be the Quantum time? 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 reduce the number of context switching, average waiting time and average turnaround time, we must find a dynamic Quantum time. This will provide a Quantum time that change automatically according to a circumstance so as to improve the general performance of the system.

Each of the proposed dynamic RR performs better when compared with static RR on the basis of number of context switching, average waiting time and average turnaround time. If that is the case, which among them should be implemented and why? In order to answer the above question, the Modified Median RR (MMRRA) was proposed.

Another problem is that, most of these algorithms do not care to include ART as a criteria for comparison. And, they do not bother much on preempting processes with negligible left over. Mis-calculation of number of context switching (NCS) is another problem to most of these algorithms.

The MMRRA would be compared against the classical RR, Improved Round Robin(IRR), Improved Mean Round Robin(IMRR), Half Life Variable Quantum Time RR(HLVQTRR) and Dynamic Round Robin with Controlled Preemption (DRRCP) using analytic and simulation techniques.

1.3 JUSTIFICATION OF THE STUDY

So far, a lot of research papers having excellent ideas on dynamic RR algorithms have been published eg IRR, HLVQTRR etc. But most of these papers only considered average waiting time, average turnaround time and number of context switching as the criteria for comparison and evaluation. They did not bother to include other criteria like multiprogramming, response time and so on.

Since dynamic RR has proven to provide better results, a new dynamic RR CPU scheduling algorithm should be developed that will provide better solutions to both the classical RR and most of the proposed dynamic RR algorithms that were reviewed. The evaluation would be done analytically and also through the use of simulation on the following criteria: average waiting time; average turnaround time; number of context switching; response time; and priority on shorter jobs. Random variable will be generated using exponential distribution function to perform the analysis.

In general, the concept of CPU scheduling would be thoroughly presented to serve as a learning resource material.

1.4 AIM AND OBJECTIVES

The aim of this work is to propose a better solution to the classical and most of the dynamic RR CPU scheduling algorithm. The basic objectives are:

1. Develop a new Dynamic RR algorithm. “Modified Median Round Robin Algorithm” (MMRRA)

2. Implement a system that will help in minimizing the overall waiting time, turnaround time, number of context switching and response time of processes.
3. Compare the proposed algorithm against five selected dynamic RR CPU scheduling algorithm.

1.5 SCOPE

This research work would focus on the scheduling algorithm for Round Robin technique. Thereby, developing a framework that will make fair consideration to shorter jobs on the queue. Many research papers with excellent ideas on RR scheduling have been published. But most of these papers focused on only average waiting time, average turnaround time and number of context switching as the criteria for comparison and evaluation.

A new dynamic RR CPU scheduling algorithm would be proposed so as to provide better solutions to both the classical RR and most of the proposed dynamic RR algorithms that were reviewed. The evaluation shall be done analytically and also through the use of simulation on the following criteria: average waiting time; average turnaround time; number of context switching; average response time and priority on shorter jobs.

1.6 METHODOLOGY

The approaches and series of steps to be taken in actualizing our research objectives are as follows:

i Random variables are generated using Exponential distribution

ii Visual basic programming language 6.0 is used in writing the program.

iii MMRRA was evaluated analytically and through simulation against the classical RR,

IRR, IMRR, DRRCP and HLVQTRR.

iv The result of each run of the simulator is also represented graphically using column graph.

v The results of the evaluation are plotted using Microsoft excel 2007.

Three data set have been used for the analysis.

i The first set has been generated by the proposed algorithm

ii The second set are the data set used in DRRCP.

iii The third set are the data set used in IMRRSJF.

Reason for using the data set in DRRCP and IMRRSJF is because MMRRA borrowed some few ideas but not exactly of these two (2) algorithms. Therefore, there is need to use their own data set for analysis.

CHAPTER TWO: LITERATURE REVIEW

2.1 THEORITICAL FRAME WORK

A literature review is an account of what has been published on a topic by accredited scholars and researchers. This chapter will give a detailed explanation of an operating system and analysis of some existing scheduling techniques.

To understand the operating system is to understand the workings of the entire computer system. Because, it is the operating system that manages each and every piece of hardware and software. To put it in a simple term, it's the software. Operating system controls every file, every device, every section of main memory, and every nanosecond of processing time. It controls who can use the system and how. In short, it's the boss (Flynn and McHoes, 2001).

The four essential managers of every operating system are: the **memory manager**, the **processor manager**, the **device manager**, and the **file manager**. In fact, these managers are the basis of all operating systems. Each manager works closely with the other managers and performs its unique role regardless of which specific operating system is being discussed.

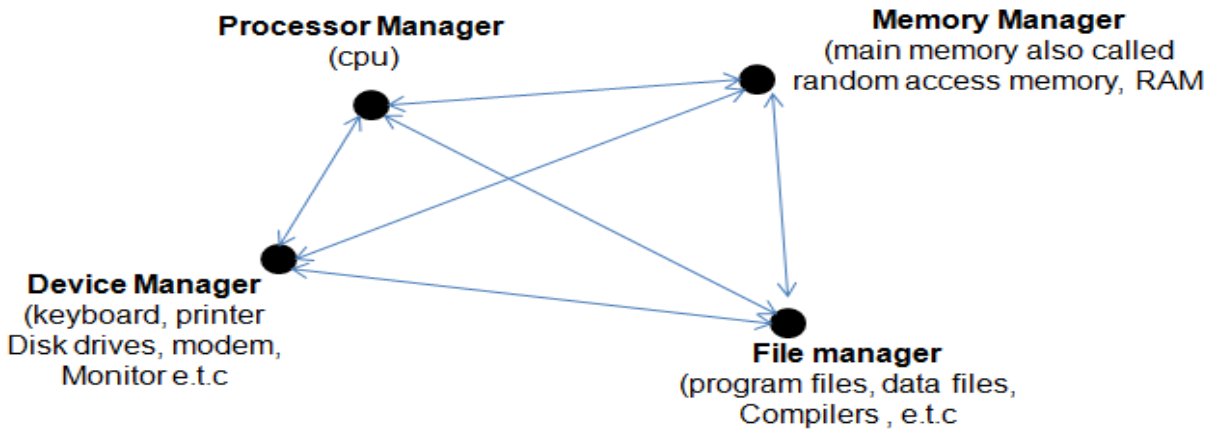

Figure 2.1: Each subsystem manager takes responsibility for its own tasks while working harmoniously with every other manager (Flynn and McHoes, 2001).

Regardless of the size or configuration of the system, each subsystem manager, shown in Figure 2.1, must perform the following tasks:

i, Monitor its resources continuously.

ii, Enforce the policies that determine who gets what, when, and how much.

iii, Allocate the resources when it's appropriate.

iv, De-allocate the resource –reclaim it-when appropriate

For instance, the memory manager is in charge of main memory, also known as RAM. It checks the validity of each request for memory space and, if it is a legal request, the memory manager allocate a portion that isn't already in use. In a multiuser environment, the memory manager set up a table to keep track of who is using which section of memory. Finally, when the time comes to reclaim the memory, the memory manager de-allocates memory.

The processor manager decides how to allocate the central processing unit (CPU). An important function of the processor manager is to keep track of the status of each process. The processor manger monitors whether the CPU is executing a process or waiting for a READ or WRITE command to finish execution. Because it handles the process' transitions from one state of execution to another, it can be compared to traffic controller. Once the processor manager allocates the processor, it sets up the necessary registers and tables and, when the job is finished or the maximum amount of time has expired, it reclaims the processor. A process is defined as an instance of execution of a program (Simon *et al*, 2014).

Conceptually, the processor manager has two levels of responsibility. One is to handle jobs as they enter the system and the other is to manage each process within those jobs. The first part is handled by the job scheduler, the high-level portion of the processor managers, which accepts or reject the incoming jobs. The second parts is handled by the process scheduler, the low-level portion of the processor Manager, which is responsible for deciding which process gets the CPU and for how long (Flynn and McHoes, 2001).

The device manager monitors every device, channel, and control unit. Its job is to choose the most efficient way to allocate all of the system devices, printers, terminals, disk drives, and so forth, based on a scheduling policy chosen by the system's designers. The device manager makes the allocation, starts its operation, and finally, de-allocates the resource.

The fourth, the file manager, keeps track of every file in the system including data files, assemblers, compilers and application programs. By using predetermined access policies, it enforces access restrictions on each file. When a file is created at the initial stage, it is declared

as either system-only, user-only, group-only, or general-access. A crucial responsibility of the operating system is to enforce these restrictions. However, it is not enough for each manager to perform its individual tasks. It must also be able to work harmoniously with every other manager (Flynn and McHoes, 2001).

2.2 MULTIPROGRAMMING

In a single user system, the processor is busy only when the user is executing a job at all other times it is idle. Processor Management in this environment is simple. However, when there are many users with many jobs on the system (this is known as a multiprogramming environment), the processor must be allocated to each job in a fair and efficient manner, which can be a complex task. A multiprogramming environment is an environment where more than one program runs (Ibrahim, 2012).

A program or job is a unit or work that has been submitted by the user. While, a process is just an executing program, including the current values of the program counter, registers, and variables. Conceptually, each process has its own virtual CPU. In reality, of course, the real CPU switches back and forth from process to process, but to understand the system, it is much easier to thing about a collection of processes running in (pseudo) parallel, than to try to keep track of how the CPU switches from program to program. This rapid switching back and forth is called multiprogramming (Tanenbaum, 2001).

Multiprogramming requires that the processor be allocated to each job or to each process for a period of time and de-allocated at an appropriate moment. If the processor is de-allocated during a program's execution, it must be done in such a way that it can be restarted later as easily as

possible. This is a delicate procedure. The Processor Manager would identify the series of events as follows:

i, Get the input for Job A	ii, Identify resources
iii, Execute the process	iv, Interrupt
v, Context switch to Job B	vi, Get the input for Job B
vii, Identify resources	viii, Execute the process
ix, Terminate Job B	x, Context switch to Job A
xi, Resume executing interrupted process	xii, Terminate Job A

As has been shown, a single processor can be shared by several jobs or several processes, but if, and only if, the operating system has a scheduling policy, as well as a scheduling algorithm, to determine when to stop working on one job and proceed to another (Flynn and McHoes, 2001)

2.3 SCHEDULING

When a computer is multiprogrammed, it frequently has multiple processes competing for the CPU at the same time. This situation occurs whenever two or more processes are simultaneously in the ready state. If only one CPU is available, a choice has to be made which process to run next. The part of the operating system that makes the choice is called the scheduler and the algorithm it uses is called the scheduling algorithm (Tanenbaum, 2001).

2.3.1 Job Scheduling Versus Process Scheduling

The Process Manager is a composite of two sub managers: one in charge of job scheduling and the other in charge of process scheduling. They're known as the Job Scheduler and the Process

Scheduler respectively. However, the scheduling of jobs is actually handled on two levels by most operating systems. Hierarchy exists between the Job Scheduler and the Process Scheduler. Jobs are initiated by the Job Scheduler based on certain criteria. Once a job is selected for execution, the Process Scheduler determines when each step, or set of steps, is executed-a decision that's also based on certain criteria. Therefore, each job (or program) passes through a hierarchy of managers. The first one it encounters is the Job Scheduler, this is also called the high-level scheduler. It is only concerned with selecting jobs from a queue of incoming jobs and placing them in the process queue, whether batch or interactive, based on each job's characteristics. The Job Scheduler's goal is to put the jobs in a sequence that will use all of the system's resources as fully as possible (Flynn and McHoes, 2001).

This is an important function. For example, if the Job Scheduler selects several jobs to run consecutively and each has a lot of I/O, then the I/O devices would be kept very busy. The CPU might be busy handling the I/O, if an I/O controller is not used, so little computation might get done. On the other hand, if the Job Scheduler selected several consecutive jobs with a great deal of computation, then the CPU would be very busy but the I/O devices would be idle waiting for I/O requests. Therefore, the Job Scheduler strives for a balanced mix of jobs that require large amounts of I/O interaction and jobs that require large amounts of computation. Its goal is to keep most components of the computer system busy most of the time.

2.4 PROCESS SCHEDULER

After a job has been placed on the READY queue by the Job Scheduler, the Process Scheduler takes over. It determines which jobs will get the CPU, when, and for how long. It also decides

when processing should be interrupted, determines which queues the job should be moved to during its execution, and recognizes when a job has concluded and should be terminated. Note that the ready queue is not necessarily a first-in, first-out (FIFO) queue. As we shall see when we consider the various scheduling algorithms, a ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list. Conceptually, however, all the processes in the ready queue are lined up waiting for a chance to run on the CPU (Silberschatz *et al.*, 2004).

The Process Scheduler is the low-level scheduler that assigns the CPU to execute the processes of those jobs placed on the ready queue by the Job Scheduler. This becomes a crucial function when the processing of several jobs has to be orchestrated. To schedule the CPU, the Process Scheduler takes advantage of a common trait among most computer programs: they alternate between CPU cycles and I/O cycles. Notice that the following job has one relatively long CPU cycle and two very brief I/O cycles It is important that the long-term scheduler select a good **process mix** of I/O-bound and CPU-bound processes. If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do. If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced. The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes. On some systems, the long-term scheduler may be absent or minimal (Silberschatz *et al.*, 2004).

```
READ A, B                    I/O cycle

C= A+ B
D = (A*B) – C                CPU cycle
E = A-B
F = D/E

WRITE A, B, C, D, E, F       I/O cycle
STOP                         terminate execution
END
```

Although the duration and frequency of CPU cycles vary from program to program, there are some general tendencies that can be exploited when selecting a scheduling algorithm. For example, I/O-bound jobs (such a printing a series of documents) have many brief CPU cycles and long I/O cycles, whereas CPU-bound jobs (such a finding the first 300 prime numbers) have long CPU cycles and shorter I/O cycles (Flynn and McHoes, 2001).

In a highly interactive environment there's also a third layer of the Processor Manager called the middle-level scheduler. In some cases, especially when the system is overloaded, the middle-level scheduler finds it is advantageous to remove active jobs from memory to reduce the degree of multiprogramming and thus allow jobs to be completed faster. The jobs that are swapped out and eventually swapped back in are managed by the middle-level scheduler. In a single-user environment, there's no distinction made between job and process scheduling because only one job is active in the system at any time. So the CPU and all other resources are dedicated to that job until it is completed (Flynn and McHoes, 2001).

2.5 STATUS JOB AND PROCESS

As a job moves through the system it's always in one of five states (or at least three). It changes from HOLD to READY, READY to RUNNING, RUNNING to WAITING and eventually to FINISHED as shown in figure 2.2.These states are called the job status or the process status.

Here's how the job status changes when a user submits a job to the system via batch to interactive mode. When the job is accepted by the system, it's put on HOLD and placed in a queue. In some systems, the job spooler (or disk controller) creates a table with the characteristics of each job in the queue and notes the important features of the job, such as an estimate of CPU time, priority, special I/O devices required and maximum memory required.

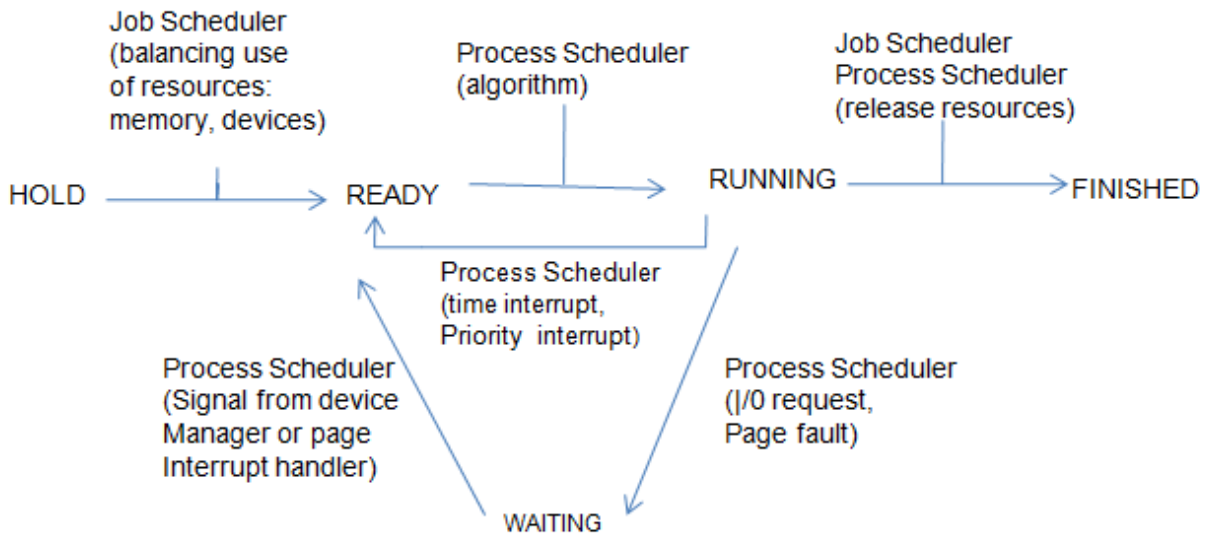

Figure 2.2: A typical job (or process) changes status as it moves through the system from HOLD to FINISHED (Flynn and McHoes, 2001).

This table is used by the Job Scheduler to decide which job is to be run next. From HOLD, the job moves to READY when it's ready to run but is waiting for the CPU. In some systems, the job (or process) might be placed on the READY list directly. RUNNING, of course, means that the job is being processed. In a single processor system this is one "job" or process. WAITTING means that the job can't continue until a specific resource is allocated or an I/O operation has finished. Upon completion, the job is FINISHED and returned to the user. The transition from

one job or process status to another is initiated by either the Job Scheduler or the Process Scheduler as follows:

i The transition from HOLD to READY is initiated by the Job Scheduler according to some predefined policy. At this point the availability of enough main memory and any requested devices are checked.

ii The transition from READY to RUNNING is handled by the process scheduler according to some predefined algorithm (i.e., FCFS, SJN, priority scheduling, SRT, or round robin, all of which will be discussed shortly).

iii The transition from RUNNING back to READY is handled by the Process Scheduler according to some predefined time limit or other criterion, for example a priority interrupt. The transition from RUNNING to WAITTING is handled by the Process Scheduler and is initiated by an instruction in the job such as a command to READ, WRITE, or other I/O request, or one that requires a page fetch.

iv The transition from WAITTING to READY is handled by the Process Scheduler and is initiated by a signal from the I/O device manager that the I/O request has been satisfied and the job can continue. In the case of a page fetch, the page fault handler will signal that the page is now in memory and the process can be placed on the READY queue.

v Eventually, the transition from RUNNING to FINISHED is initiated by the Process Scheduler or the Job Scheduler either when (1) the job is successfully completed and it

ends execution or (2) the operating system indicates that an error has occurred and the job is being terminated prematurely (Flynn and McHoes, 2001).

.

2.6 PROCESS CONTROL BLOCKS

Each process in the system is represented by a data structure called a Process Control Block (PCB) that performs the same function as a traveler's passport. The PCB (illustrated in figure 2.3) contains the basic information about the job including what it is, where it's going, how much of its processing has been completed, where it's stored, and how much it has spent in using resources. PCB is also called a *task control block* (Silberschatz *et al*, 2004).

.

2.6.1 Process Identification

Each job is uniquely identified by the user's identification and a pointer connecting it to its descriptor (supplied by the Job Scheduler when the job first enters the system and is placed on HOLD).

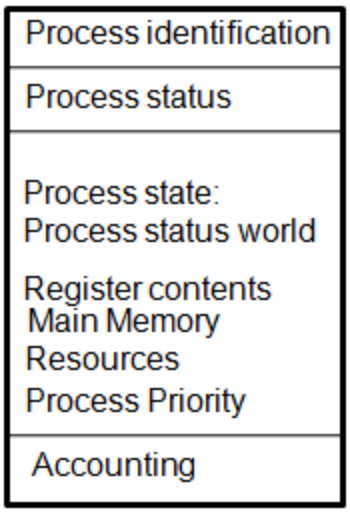

Figure 2. 3: Contents of each job's process control block (Flynn and McHoes, 2001).

2.6.2 Process Status

This indicates the current status of the job-HOLD, READY, RUNNING, OR WAITING – and the resources responsible for that status.

2.6.3 Process State

This contains all of the information needed to indicate the current state of the job such as:

i *Process Status Word*- the current instruction counter and register contents when the job isn'trunning but is either on HOLD, READY or WAITING. If the job is running, this information is left undefined.

ii *Register Contents*-the contents of the register if the job has been interrupted and is waiting to resume processing.

iii *Main Memory*- pertinent information, including the address where the job is stored and, in the case of virtual memory, the mapping between virtual and physical memory locations.

iv *Resources*- information about all resources allocated to this job. Each resource has an identification file listing its type and a field describing details of its allocation such as the sector address on a disk. These resources can be hardware units (disk drives or printers, for example) or files.

v *Process Priority*-used by systems using a priority scheduling algorithm to select which job will be run next.

2.6.4 Accounting

This contains information used mainly for billing purposes and performance measurement it indicates what kind of resources the job used and for how long. Typical charges include:

i Amount of CPU time used from beginning to the end of its execution.

ii Total time the job was in the system until it exited.

iii Main storage occupancy-how long the job stayed in memory until it finished execution. This is usually a combination of time and space used; for example, in a paging system it may be recorded in units of page/seconds.

iv Secondary storage used during execution. This, too, is recorded as a combination of time and space used.

v System programs used such as compilers, editors, or utilities.

vi Number and type of I/O operations, including I/O transmission time, that includes utilization of channels, control units, and devices.

vii Time spent waiting for I/O completion.

viii Number of input records read (specifically, those entered on-line or coming from optical scanners, card readers, or other input devices), and number of output records written

(specifically, those sent to the line printer). This last one distinguishes between secondary storage devices and typical I/O devices (Flynn and McHoes, 2001).

.

2.7 PCBs AND QUEUING

A job's PCB is created when the Job Scheduler accepts the job and is updated as the job progresses from the beginning to the end of its execution. Queues use PCBs to track jobs the same way customs officials use passports to track international visitors. The PCB contains all of the data about the job needed by the operating system to manage the processing of the job. As the job moves through the system its progress is noted in the PCB.

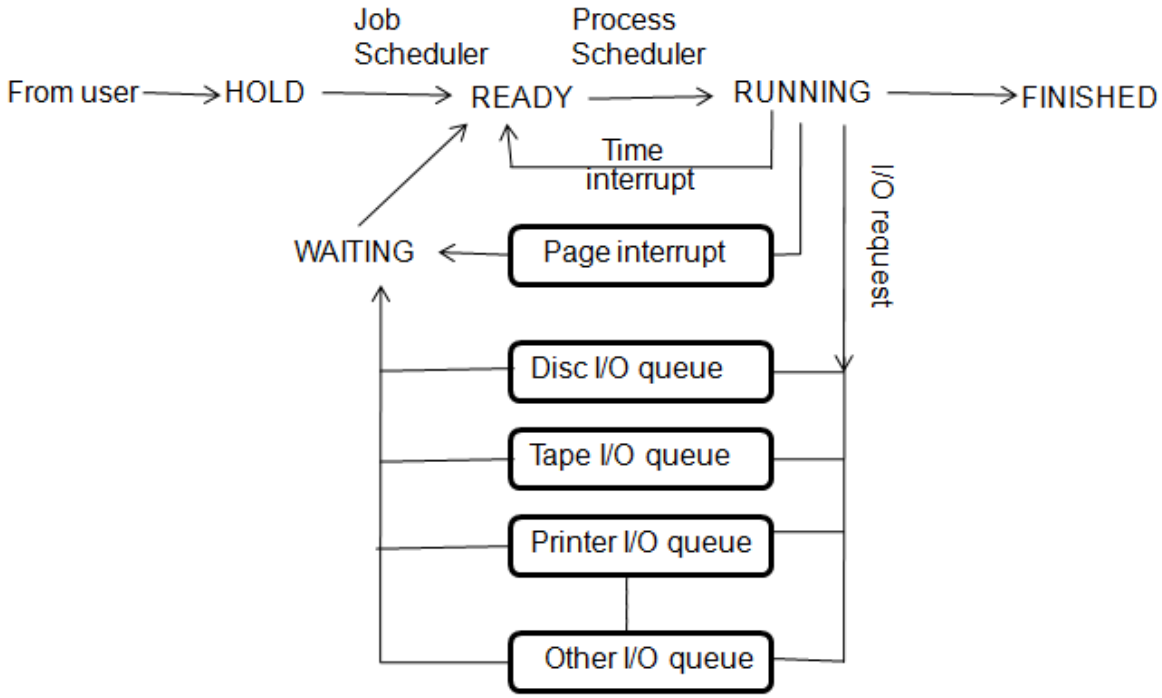

Figure 2. 4: Queuing paths from HOLD to FINISHED. The job and processors schedulers release the resources when the job leaves the RUNNING state(Flynn and McHoes, 2001).

The PCBs, not the jobs, are linked to form the queues as shown in Figure2.4. Although each PCB is not drawn in detail the reader should imagine each queue as a linked list of PCBs. The

PCBs for every ready job are linked on the READY queue, and all of the PCBs for the jobs just entering the system are linked on the HOLD queue. The jobs that are waiting, however, are linked by "reason for waiting," so the PCBs for the jobs in this category are linked into several queues. For example, the PCBs for jobs that are waiting for I/O on a specific disk drive are linked together, while those waiting for the line printer are linked in a different queue. These queues need to be managed in an orderly fashion and that's determined by the process scheduling policies and algorithms (Flynn and McHoes, 2001).

.

2.8 PROCESS SCHEDULING POLICIES

In a multiprogramming environment there are usually more jobs to be executed than could possibly be run at one time. Before the operating system can schedule them, it needs to resolve three limitations of the system:

i there are a finite number of resources (such as disk drives, printers, and tape drives);

ii some resources, once they are allocated, can not be shared with another job (e.g., printers)

iii some resources require operator intervention-that is, they can't be reassigned automatically from job to job (such as tape drives) (Flynn and McHoes, 2001).

.

What's a good process scheduling policy? Several criteria come to mind, but notice in the list below that some contradict each other:

I Maximize throughput-run as many jobs as possible in a given amount of time. This could be accomplished easily by running only short jobs or by running jobs without interruptions.

ii Minimize response time-quickly turn around interactive requests. This could be done by running only interactive jobs and letting the batch jobs wait until the interactive load ceases.

iii Minimize turnaround time- move entire jobs in and out of the system quickly. This could be done by running all batch jobs first (because batch jobs can be grouped to run more efficiently than interactive jobs).

iv Minimize waiting time-move jobs out of the ready queue as quickly as possible. This could only be done by reducing the number of users allowed on the system so the CPU would be available immediately whenever a job entered the ready queue.

v Maximize CPU efficiency-keep the CPU busy 100percent of the time. This could be done by running only CPU-bound jobs (and not l/O-bound jobs).

vi Ensure fairness for all jobs-give everyone an equal amount of CPU and l/O time. This could be done by not giving special treatment to any job; regardless of its processing characteristics or priority.

As we can see from the above list, if the system favors one type of user then it hurts another or does not efficiently use its resources. The final decision rests with the system designer, who must determine which criteria are most important for that specific system. For example, you might decide to "maximize CPU utilization while minimizing response time and balancing the use of all system components through a mix of l/O-bound and CPU-bound jobs. " So you would select

the scheduling policy that most closely satisfied your criteria. Although the Job Scheduler selects jobs to ensure that the READY and l/O queues remain balanced, there are instances when a job claims the CPU for a very long time before issuing an l/O request. If l/O requests are being satisfied (this is done by an l/O controller). This extensive use of the CPU will build up the ready queue while emptying out the l/O queues which create an unacceptable imbalance in the system (Flynn and McHoes, 2001).

.

To solve this problem, the Process Scheduler often uses a timing mechanism and periodically interrupts running processes when a predetermined slice of time has expired. When that happens, the Scheduler suspend all activity on the Job currently running and reschedules it into the READY queue; which will be continued later. The CPU is now allocated to another job that runs until one of the three things happens: i, the timer goes off (the job moves to the READY queue), ii, the job issues an I/O command (the job moves to the WAIT queue), or iii, the job is finished (the FINISHED queue).In multi-programming environments, an I/O request is called a natural wait (it allows the processor to be allocated to another job). The objective of multiprogramming is to maximize CPU utilization (Silberschatz *et al,* 2004).

.

.A scheduling strategy that interrupts the processing of a job and transfers the CPU to another job is called a preemptive scheduling policy; it is widely used in time-sharing environments. The alternatives, of course, is a non-preemptive scheduling policy, which functions without external interrupts (interrupts external to the job). Therefore, once a job captures the processor and begins execution, it remains in the running state uninterrupted until it issues an l/O request (natural wait)

or until it is finished (with exceptions made for infinite loops, which are interrupted by both preemptive and non-preemptive policies). (Flynn and McHoes, 2001).

2.9 PROCESS SCHEDULING ALGORITHMS

The Process Scheduler relies on a process scheduling algorithm, based on a specific policy; to allocate the CPU and move jobs through the system. Early operating systems used non-preemptive policies designed to move batch jobs through the system as efficiently as possible. Most current systems, with their emphasis on interactive use and response time, use an algorithm that takes care of the immediate requests of interactive users. Here are six process scheduling algorithms that have been used extensively (Flynn and McHoes, 2001)..

2.9.1 First-Come First Served

First-come, first-served (FCFS) is a non-preemptive scheduling algorithm that handles jobs according to their arrival time: the earlier they arrive, the sooner they're served. It's very simple algorithm to implement because it uses a FIFO queue. This algorithm is fine for most batch systems, but it is unacceptable for interactive systems because interactive users expect quick response times. with FCFS, as a new job enters the system its PCB is linked to the end of the READY queue and it is removed from the front of the queue when the processor becomes available- that is, after it has processed all of the jobs before it in the queue. In a strictly FCFS system there are no WAIT queues (each job is run to completion), although there may be systems in which control (context) is switched on a natural wait (I/O request) and then the job resumes on I/O completion. The following examples presume a strictly FCFS environment (no multiprogramming).Turn-around time is unpredictable with the FCFS policy. Consider the following three jobs:

i Job A has a CPU cycle of 15 milliseconds

ii Job B has a CPU cycle of 2 milliseconds

iii Job C has a CPU cycle of 1 milliseconds

For each job, the CPU cycle contains both the actual CPU usage and the I/O requests. That is, it is the total run time. Using a FCFS algorithm with an arrival sequence of A, B, C the timeline (Gantt chart) is shown in figure 2.5

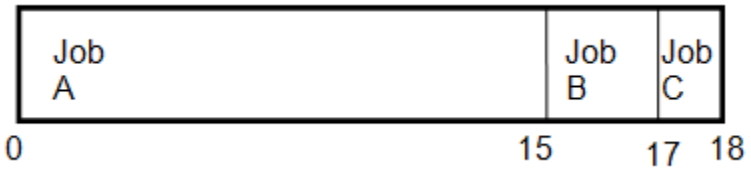

Figure 2.5: Timeline for job sequence A,B,C using the FCFS algorithm (Flynn and McHoes, 2001).

If all three jobs arrive almost simultaneously, we can calculate that the turnaround time for job A is 15, for job B is 17, and for job C is 18. So the average turnaround time is

$\frac{15+17+18}{3} = 16.67$

However, if the jobs arrived in a different order, say C, B, A then the result using the same FCFS algorithm would be as shown in Figure 2.6

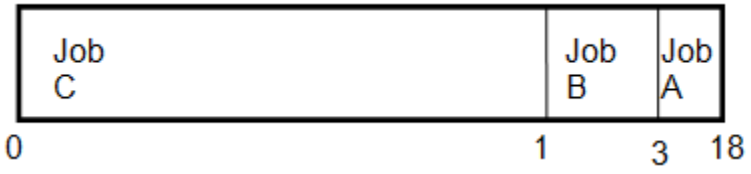

Figure 2. 6: Timeline for job sequence C,B,A using the FCFS algorithm (Flynn and McHoes, 2001).

In this example, the turnaround time for job A is 18, for job B is 3, and for job C is 1 and the average turnaround time is $\frac{18+3+1}{3} = 7.3$

That is quite an improvement over the first sequence. Unfortunately, these two examples illustrate the primary disadvantage of using the FCFS concept-the average turnaround times vary widely and are seldom minimized. In fact, when there are three jobs in the READY queue, the system has only a 1 in 6 chance of running the jobs in the most advantageous sequence (C, B, A). with four jobs the odds fall to 1 in 24 and so on. If one job monopolizes the system, the extent of its overall effect on system performance depends on the scheduling policy and whether the job is CPU-bound or l/O-bound. FCFS is a less attractive algorithm than one that would serve the shortest job first, as the next scheduling algorithm doe (Flynn and McHoes, 2001)..

2.9.2 Shortest Job Next

Shortest job next (SJN) is a non preemptive scheduling algorithm (also known as shortest job first or SJF) that handles jobs based on the length of their CPU cycle time. It's easiest to implement in batch environments where the estimated CPU time required to run the job is given in advance by each user at the start of each job. However, it doesn't work in interactive systems because users don't estimate in advance the CPU time required to run their jobs.

For example, here are four batch jobs, all in the READY queue, for which the CPU cycle, or run time, is estimated as follows:

Job	a	b	c	d
CPU cycle	5	2	6	4

The SJN algorithm would review the four jobs and schedule them for processing in this order: B, D, A, C. the timeline is shown in Figure 2.7

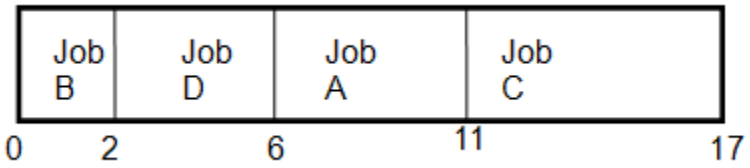

Figure 2. 7: Timline for job sequence B, D, A, C using the SJN algorithm (Flynn and McHoes, 2001)

The average turnaround time is $\frac{2+6+11+17}{4} = 9.0$

Let's take a minute to see why this algorithm can be proved to be optimal and will consistently give the minimum average turnaround time. We will use the previous example to derive a general formula. If we look at Figure 2.7, we can see that job B finished in its given time (2), job D finishes in its given time plus the time it waited for B to run (4 + 2), Job A finishes in its given time plus D's time plus B's time (5 + 4 + 2), and job C finishes in its given time plus that of the previous three (6 + 5 + 4+2). So when calculating the average, we have.

$$\frac{[(2)+(4+2)+(5+2+4)+(6+5+4+2)]}{4} = 9.0 \ldots\ldots\ldots \text{equation 2.1}$$

As you can see, the time for the first job appears in the equation four times-once for each job. Similarly, the time for the second job appears three times (the number of jobs minus one). The time for the third job appears twice (number of jobs minus 2) and the time for the fourth job appears only ones (number of jobs minus 3).

So the above equation can be rewritten as $\frac{[4*2+3*4+2*5+1*6]}{4} = 9.0$

Because the time for the first job appears in the equation four times, it has four times the effect on the average time than does the length of the fourth job, which appears only once. Therefore, if

the first job requires the shortest computation time, followed in turn by the other jobs, ordered from shortest to longest, then the result will be the smallest possible average. The formula for the average is as follows:

$$\frac{[t1\ (n) + t2\ (n-1) + t3\ (n-2) + \cdots + tn\ (1)]}{n} \quad \text{……..equation 2.2}$$

Where n is the number of jobs in the queue and t_j (j = 1, 2, 3, …., n) is the length of the CPU cycle for each of the jobs. However, the SJN algorithm is optimal only when all of the jobs are available at the same time and the CPU estimates are available and accurate (Flynn and McHoes, 2001).

2.9.3 Priority Scheduling

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems, even though it may give slower turnaround to some users. This algorithm givens preferential treatment to important jobs. It allows the programs with the highest priority to be processed first, and they aren't interrupted until their CPU cycles (run times) are completed or a natural wait occurs. If two or more jobs with equal priority are present in the ready queues, the processor is allocated to the one that arrived first (first-come first-served within priority). Priorities can be assigned by a system administrator using characteristics extrinsic to the jobs. For example, they can be assigned based on the position of the user (researchers first, students last) or, in commercial environments, they can be purchased by the users who pay more for higher priority to guarantee the fastest possible processing of their jobs. With a priority algorithm, jobs are usually linked to one of several READY queues by the Job Scheduler based on their priority so the Process Scheduler manages multiple READY queues

instead of just one. Priorities can also be determined by the Processor Manager based on characteristic intrinsic to the jobs such as:

i Memory requirements. Jobs requiring large amounts of memory could be allocated lower priories than those requesting small amounts of memory, or vice versa.

ii Number and type of peripheral devices. Jobs requiring many peripheral devices would be allocated lower priorities than those requesting fewer device.

iii Total CPU time. Jobs having a long CPU cycle, or estimated run time, would be given lower priorities than those having a brief estimated run time.

iv Amount of time already spent in the system. This is the total amount of elapsed time since the job was accepted for processing. Some systems increase the priority of jobs that have been in the system for an unusually long time to expedite their exit. This is known as aging.

These criteria are used to determine default priorities in many systems. The default priorities can be overruled by specific priorities named by users (Flynn and McHoes, 2001).

2.9.4 Shortest Remaining Time

Shortest remaining time (SRT) is the preemptive version of the SJN algorithm. The processor is allocated to the job closest to completion-but even this job can be preempted if a newer job in the READY queue has a time to completion that's shorter. This algorithm can't be implemented in an interactive system because it requires advance knowledge of the CPU time required to finish

each job. It is often used in batch environments, when it is desirable to give preference to short jobs, even though SRT involves more overhead than SJN because the operating system has to frequently monitor the CPU time for all the jobs in the READY queue and must perform context switching for the jobs being swapped. The example in Figure 2.8 shows how the SRT algorithm works with four jobs that arrived in quick succession (they have 1 CPU cycle apart).

Arrival time:	0	1	2	3
Job:	A	B	C	D
CPU cycle:	6	3	1	4

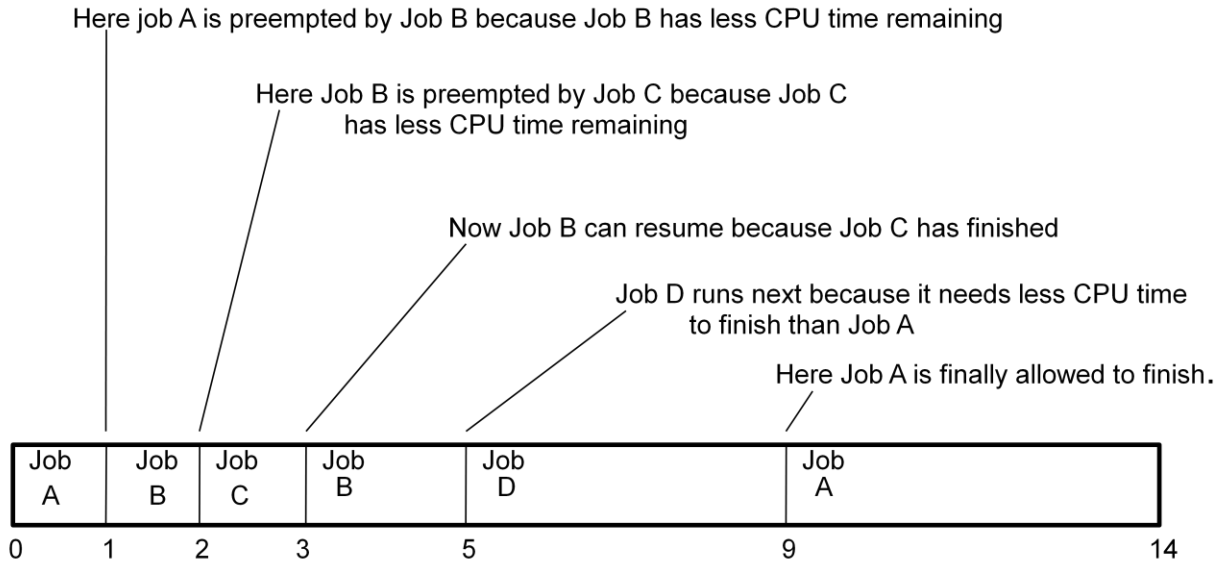

Figure 2. 8: Timeline for job sequence A, B, C, D using the preemptive SRT algorithm. Each job is interrupted after one CPU circle if another job is waiting with less CPU time remaining (Flynn and McHoes, 2001)

In this case the turnaround time is the completion time of each job minus its arrival time:

Job:	A	B	C	D
Turnaround:	14	4	1	6

So the average turnaround time is $\frac{14+4+1+6}{4} = 6.25$

How does that compare to the same problem using the non-preemptive SJN policy? Figure 2.9 shows the same situation using SJN (Flynn and McHoes, 2001).

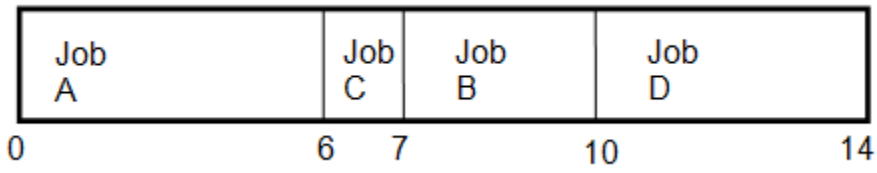

Figure 2. 9: Timeline for the same job sequence A, B, C, D using the non-preemptive SJN algorithm (Flynn and McHoes, 2001)..

In this case the turnaround time is

Job:	A	B	C	D
Turnaround:	6	9	5	11

So the average turnaround time is $\frac{6+9+5+11}{4} = 7.75$

Note in Figure 2.9 that initially A is the only job in the READY queue so it runs first and continues until it's finished because SJN is a non-preemptive algorithm. The next job to be run is C because when Job A is finished (at time 6), all of the other jobs (B, C, and D) have arrived. Of those three, C has the shortest CPU cycle, so it is next one to run, then B. and finally D.

Therefore, with this example, SRT at 6.25 is faster than SJN at 7.75. However, we neglected to include the time required by the SRT algorithm to do the context switching. Context switching is required by all preemptive algorithms. When job A is preempted all of its processing information must be saved in its PCB for later use and, the contents of job B's PCB are loaded into the

appropriate registers so that it can start running . This is a context switch; later, when Job A is once again assigned to the processor, another context switching is performed. How the context switching is actually done depends on the architecture of the CPU; in many systems there are special instructions that provide quick saving and restoring of information. The switching is designed to be performed efficiently but, no matter how fast it is, it still takes valuable CPU time. So although SRT appears to be faster, in a real operating environment its advantages are diminished by the time spent in context switching (Flynn and McHoes, 2001).

A precise comparison of SRT and SJN would have to include the time required to do the context switching. Context-switch time is pure overhead and is highly dependent on hardware support (Silberschatz *et al,* 2004).

2.9.5 Round Robin

The round robin is a scheduling algorithm that assigns fixed time-slices called 'Quantum' to processes as they are scheduled. Each process in the queue is allocated the CPU through the amount of Quantum set by the designer (Rakesh, 2011). .Round robin is a preemptive process scheduling algorithm that is used extensively in interactive systems because it's easy to implement. If Jobs are placed in the READY queue using a first-come first-served scheme, The process scheduler selects the first job from the front of the queue, sets the timer to the time quantum, and allocates the CPU to this job. If processing isn't finished when time expires, the job is preempted and put at the end of the READY queue and its information is saved in its PCB. In the event that the job's CPU cycle is shorter than the time quantum, one of two actions will take place:

i if this is the job's last CPU cycle and the job is finished, then all resources allocated to it are released and the completed job is returned to the user;

ii if the CPU cycle has been interrupted by an I/O request, then information about the job is saved in its PCB and it is linked at the end of the appropriate I/O queue. Later, when the I/O request has been satisfied, it is returned to the end of the READY queue to await allocation of the CPU.

It is generally referred to as starvation-free algorithm; where each process is considered continuously until it finishes its available burst-time. RR has the advantage of being easy to implement, each process gets equal share of the CPU and CPU utilization is almost 100% as the CPU never gets idle (Ibrahim, 2012).

The example in figure 2.10 illustrates a round robin algorithm with a time slice of 4 milliseconds (I/O requests are ignored):

Arrival time:	0	1	2	3
Job:	A	B	C	D
CPU cycle:	8	4	9	5

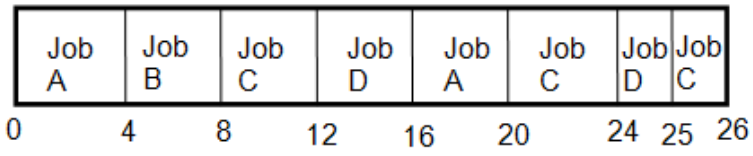

Figure 2.10: Timeline for job sequence A, B, C, D using the preemptive Round Robin algorithm with time slices of 4ms (Flynn and McHoes, 2001).

The turnaround time is the completion time minus the arrival time:

Job:	A	B	C	D
Turnaround:	20	7	24	22

So the average turnaround time is $\frac{20+7+24+22}{4} = 18.25$

Note that is Figure 2.10 Jobs A was preempted once because it needed eight milliseconds to complete its CPU cycle, while job B terminated in one time quantum. Job C was preempted twice because it needed nine milliseconds to complete its CPU cycle, and job D was preempted once because it needed five milliseconds. In their last execution or swap into memory, both jobs D and C used the CPU for only one millisecond and terminated before their last time quantum expired, releasing the CPU sooner. The efficiency of round robin depends on the size of the time quantum in relation to the average CPU cycle. Setting the quantum too short causes too many process switches and lowers the CPU efficiency, but setting it too long may cause poor response to short interactive requests. A quantum around 20-50msec is often a reasonable compromise (Tanenbaum, 2001).

Consider the three examples in figure 2.11. Job A has a CPU cycle of eight milliseconds. The amount of context switching increases as the time quantum decreases in size.

In Figure 2.11 the first case (a) has a time quantum of 10 milliseconds and there is no context switching (and no overhead). The CPU cycle ends shortly before the time quantum expires and the job runs to completion. For this job with this time quantum, there is no difference between the round robin algorithm and the FCFS algorithm.

In the second case (b), with a time quantum of five milliseconds, there is one context switch. The job is preempted once when the time quantum expires, so there is some overhead for context switching and there would be a delayed turnaround based on the number of other jobs in the system.

In the third case (c), with a time quantum of one millisecond, there are seven context switches because the job is preempted every time the time quantum expires; overhead becomes costly and turnaround time suffers accordingly.

What is the best time quantum size? The answer should be predictable by: i, it depends on the system. If it's an interactive environment, the system is expected to respond quickly to its users, especially when they make simple requests. ii, If it is a batch system, response time is not a factor (turnaround is) and overhead becomes very important.

Here are two general rules of thumb for selecting the proper time quantum: (1) it should be long enough to allow 80 percent of the CPU cycles to run to completion, and (2) it should be at least 100 times longer than the time required to perform one context switch. These rules are used in some systems, but they are not inflexible (Finkel, 1986).

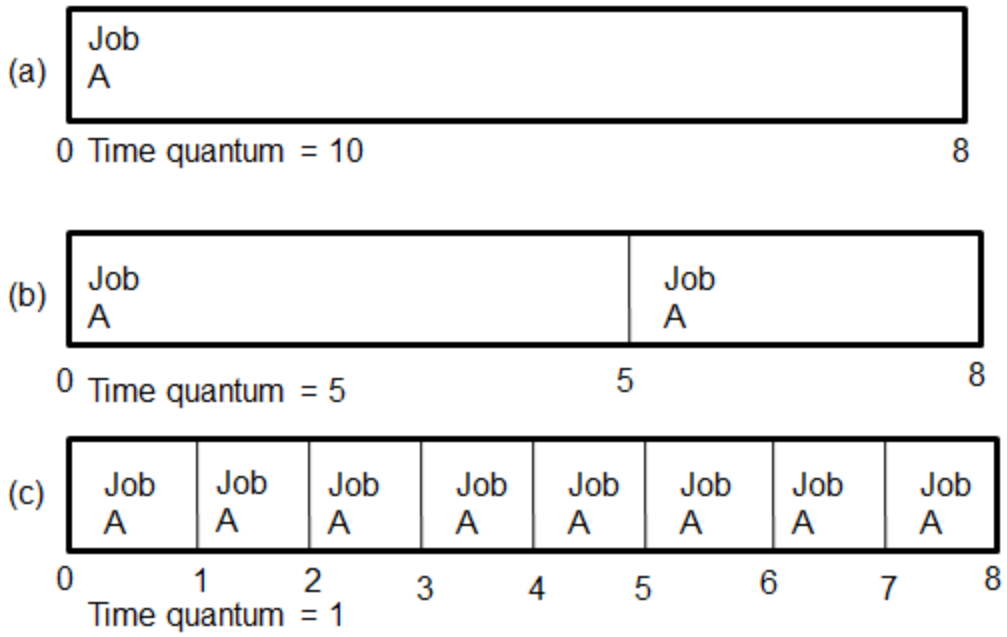

Figure 2. 11: Context switches for job A with three different time quantum. In (a), the job finishes before the time quantum expires. In (b) and (c), the time quantum expires first, interrupting the job (Flynn and McHoes, 2001).

2.10 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 time in the ready queue and remain constant throughout the entire procedure. The static nature made it difficult 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. Dynamic RR proven to be better as compared with the classical RR on the basis of average waiting time, average turnaround time and number of context switching (Flynn and McHoes, 2001).

2.11 CONTEXT SWITCHING

A context switch (also sometimes referred to as a process switch or a task switch) is the switching of the central processing unit (CPU) from one process or thread to another. A process (also sometimes referred to as a task) is an executing (i.e., running) instance of a program. In Linux, threads are lightweight processes that can run in parallel and share an address space (i.e., a range of memory locations) and other resources with their parent processes (i.e., the processes that created them). A context is the contents of a CPU's registers and program counter at any point in time. A register is a small amount of very fast memory inside of a CPU (as opposed to the slower RAM main memory outside of the CPU) that is used to speed the execution of computer programs, by providing quick access to commonly used values, generally those in the midst of a calculation. A program counter is a specialized register that indicates the position of the CPU in its instruction sequence and which holds either the address of the instruction being executed or the address of the next instruction to be executed, depending on the specific system. Context switching can be described in slightly more detail as the kernel (i.e., the core of the operating system) performing the following activities with regard to processes (including threads) on the CPU:

i suspending the progression of one process and storing the CPU's state (i.e., the context) for that process somewhere in memory,

ii retrieving the context of the next process from memory and restoring it in the CPU's registers

iii returning to the location indicated by the program counter (i.e., returning to the line of code at which the process was interrupted) in order to resume the process. (Linfo, 2006)

The Cost of Context Switching: Context switching is generally computationally intensive. That is, it requires considerable processor time, which can be on the order of nanoseconds for each of the tens or hundreds of switches per second. Thus, context switching represents a substantial cost to the system in terms of CPU time and can, in fact, be the most costly operation on an operating system.

Consequently, a major focus in the design of operating systems has been to avoid unnecessary context switching to the extent possible. However, this has not been easy to accomplish in practice (Linfo, 2006).

2.12 REVIEW OF VARIOUS DYNAMIC RR CPU SCHEDULING TECHNIQUE

The following dynamic RR CPU scheduling algorithms were reviewed:

In the work of Behera *et al.*, (2012), they adopted a method of finding the dynamic time quantum by taking the mean of burst time of the processes and fills the Ready Queue according to arrival time. They calculate the Response ratio of each process: Response Ratio = (Waiting time + Service Time)/ Service Time and assign the CPU to the process with Highest Response Ratio. After the process is being assigned to the CPU again the Response Ratio is calculated with the updated waiting time of the processes. This loop is continued until all the processes are being executed by the CPU. The dynamic time quantum is computed by taking the mean of the remaining burst time. The processes with shorter burst time and higher waiting time are executed first resulting in better turnaround time and better waiting time.

In the same vein, Mishra and Khan**(2012),** tried to show how the improve RR works without drifting away from the original RR, in their work, the algorithm picks the first process from the ready queue and allocate the CPU to it for a time interval of up to 1 time quantum. After completion of process's time quantum, it checks the remaining CPU burst time of the currently running process. If the remaining CPU burst time of the currently running process is less than 1 time quantum, the CPU again allocated to the currently running process for remaining CPU burst time. In this case this process will finish execution and it will be removed from the ready queue. The scheduler then proceeds to the next process in the ready queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than 1 time quantum, the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

Also, in reference to the work of Behera *et al.*, (2010), processes on the ready queue are arranged in ascending order according to their burst time. Then time quantum is calculated. For finding an optimal time quantum, median method is followed. The median can be find out using the following formulae.

$$\text{Median } \overline{x} = \begin{cases} Y_{n+2/2} & \text{if n is odd} \\ 1/2\,[Y_{n/2} + Y_{1+n/2}\,] & \text{if n is even} \end{cases}$$

Where, $\overline{x}$ = median

y = number located in the middle of a group of numbers arranged in ascending order

n = number of processes

Here, the time quantum is assigned to the processes. This time quantum is recalculated taking the remaining burst time in account after each cycle.

In another work by Shyam and Nandal (2014), in their approach, their proposed algorithm used round robin with shortest Job first scheduling. In this approach, all processes arrived at time 0 and, all processes present in the ready queue are sorted in ascending order of their burst time. Time Quantum (TQ) was calculated using the formula :

Time Quantum = Square root of (mean of the remaining burst time * Highest Burst)

$TQ = \sqrt{((1/n)\ (A1+ A2 + \ldots + An)*An)}$ where

n = Number of processes on the queue

A1, A2,….An = Burst time of the processes on the queue

An = A process with the highest burst time on the queue.

And if TQ > maximum burst time, then max(BT) → TQ

In each round, the new time quantum is calculated using the same formula. Which they later concluded that their approach performs better.

In the work of Noon *et al.,* (2011),they stated that, the concept of this algorithm is to take the average of the processes CPU burst at each cycle and use it as the quantum in 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. This method needs two registers to be identified:

SR: Register to store the sum of the remaining burst time in the ready queue.

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.

Another work by Banerjee *et al.*,(2012a), they proposed a method of finding quantum time as the mean of the summation of the average and the maximum burst time. That is;

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

Quantum time =(AVG + Maximum Burst)/2

The value of the quantum time is dynamically adjusted as process execution progresses. That is to say, after the first round, the same procedure is followed in obtaining a new quantum time.

In another review by Mohanty *et al.*,(2011), they adopted a method by executing the processes according to the new calculated Fit Factor 'f' and using the concept of dynamic time quantum. "f" is calculated for each process. The process having the lowest "f" value will be scheduled first. The two important criteria that decide the early execution of processes are – higher user priority and shorter burst time. As user priority has higher importance than other factors, so it is given a weight age of 60% and burst time is given 40%, assuming that all the processes have same arrival time i.e. arrival time = 0. To calculate the fit factor, they use the following criteria:

User Priority = UP

User Priority Weight = UW

Shorter Burst time Priority = SP

Burst time Priority Weight = BW.

Then Fit Factor "f" can be calculated as:

f = UP * UW + SP * BW---------- (1)

To get the optimal time quantum, median of the remaining burst time is taken as the time quantum. The approach compared the performance of Fittest job First Dynamic Round Robin

(FJFDRR) andPriority Based Static Round Robin (PBSRR) which they later concluded their approach performs better.

In the work of Banerjee *et al.,* (2012b), they stated that: The burst time of the processes is taken as sorted increasing order so that it will give better turnaround time and waiting time. Generally in Round Robin algorithm the performance depends upon the size of fixed or static Time Quantum (TQ). If TQ is too large then Round Robin algorithm approximate to First Come First Served (FCFS). If the Time Quantum is too small then there will be many context switching between the processes. They further stated that their approach solved this problem by taking a dynamic TQ. Where TQ is the Average of the summation of Mid and Max process.

Mid=(Min + Max)/2 TQn=(Mid+ Max)/2

The time Quantum is re-calculated in every round. They further stated that if there is only one process in a particular round, the time Quantum will take the process burst time.

In reference to the work of Simon *et al.,* (2014a), they Looked at a way of solving unnecessary context switching applied on preemption to processes that are using the CPU in RR scheduling as much as possible, DRRCP tries not to ruin the basic RR idea but to improve on its preemptive technique. The classical RR uses the average of processes in the ready queue as the quantum time, so is DRRCP. The only addition to classical RR is avoidance to unnecessary preemption of processes. This is the reason why this technique has a dynamic QT as opposed to the classical RR which has a static QT. In their work, they believe that a process should not be preempted with left over job of 5percent or less having completed 95 percent of its job. But this is not the case with classical RR, it always incur unnecessary preemption cost which can be avoided. The

idea is that, if the quantum time allows a process to execute up to 95 percent of its job, it should be allowed to execute its left over job without being preempted. In this scheduling, a process using the CPU may not be preempted unless its quantum time is exhausted and the remaining of its burst time is more than 5% .

On the other hand, if the quantum time finishes and greater than or equal to 95percent of the job is processed, that process should be allowed to run to completion, otherwise, it should be preempted. They further stated that, this technique can be applied to all RR algorithms, classical or dynamic. It gives priority to processes using the CPU while having a smaller left overtime of 5 percent or less to run to completion. The dynamism has to do with allowing a process not to be preempted unnecessarily. As long as 95 percent of a process job is executed, quantum time increases automatically from the average value to its actual CPU burst.

Another work of Simon *et al.,* (2014b), stated that sometimes, some processes may go for more than two round and as such increases the number of context switching, average waiting time and average turnaround time. But HLVQTRR ensures that half of every processes' bursts be executed in the first round, and in the second round the remaining half should run to completion. This will greatly improve multiprogramming irrespective of the variation in the processes' length in the ready queue. In SJF, the major problem is starvation. This is solved because only half of a process's job is executed what is left is preempted and the next process is attended to and so on. The major emphasis of this scheduling are: high multiprogramming, high response time, simplicity, elimination of unnecessary context switching and fairness which are the main criteria of RR CPU scheduling algorithm. At least, whatever the case may be, response time for each process will be high. This attribute is of high interest to real time systems.

Another advantage of HLVQRR is that it maximizes the cost of calculating quantum time 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. Other algorithms may go for second and third round with just two or three processes out of many. This may cause a serious disadvantage to RR scheduling algorithms. .

CHAPTER THREE: DESIGN OF A MODIFIED MEDIAN ROUND ROBIN ALGORITHM

3.1 INTRODUCTION

After going through various literature on the topic at hand, I came across two literature that are quite relevant to my research. They are, DRRCP proposed by (Simon et al., 2014) and IMRRSJF proposed by (Shyam and Nandal, 2014). In DRRCP, the calculation of QT is based on 95% of the process CPU burst time. If the 95% is less than or equal to the quantum time, it should run to completion otherwise, it should be preempted if its time slice expired (the purpose is to reduce unnecessary preemption). IMRRSJF on the other hand uses the formula √(mean * highest burst) as its QT (the purpose is to achieve optimal performance)

The approach in this work combines some similar properties of these two algorithms DRRCP and IMRRSJF to make up the proposed algorithm MMRRA. The first consideration on this work is the generation of processes and their burst time through a distribution. The QT is calculated using the formula QT = √ (median * HB). These processes that have been generated will be assign to the CPU using shortest job first approach. Also, a condition is made in such a way that if a process finishes its time quantum and the remaining burst time of the process is greater 10% of its burst time, it will go for the second round else it will be allow to run to completion.

In order to put all these ideas together, there will be need to have a comprehensive evaluation and comparison (using Gantt chart base on the general process properties: AWT, ATT, ART and NCS) of the proposed algorithm MMRRA against five previous literatures viz: Classical RR, An Improved RR(IRR), Improved mean RR with shortest job first (IMRRSJF), Half life variable

quantum time RR (HLVQTRR) and Dynamic round robin with controlled preemption (DRRCP). All these will be designed by some algorithms and tested as implementation using VB.

3.2 MMRRA ALGORITHM

Below is the algorithm for the proposed work, the Modified Median Round Robin (MMRRA).

1. //r = Number of processes
   //Pi= ith Process
   //i = Loop variable
   //MTQ = quantum time
   //BT= Process burst time
   //RBT = Remaining burst time of process
   // MD= Median
   // PCT = Threshold percent
2. //calculate the quantum time (MTQ)
   //MTQ = square root of (median*highest burst)
   // calculate median
   Q = {P1, P2, ……..,Pr} // sorted processes
    If (r is odd) then
      MD = BT (r+1)/2
else
      MD = (BT r/2 + BT (r/2+1))/2
end If
    MTQ = √(MD * BTr)
3. //Assign MTQ to (1 to r) processes
       For (i = 1; i<= Q.length; i++);
         $BT_i = P_i$ .get BT();
           t = BT_i- MTQ
            If (t<0) then
              $RBT_i = 0$
             Else
              $RBT_i = t$;
         If (RBT_i<= ((BT_i*PCT)/100) then
Schedule(P_i , BT_i);
          Else
Schedule(P_i, MTQ);

Q.append (P_i);
4. // Assign MTQ to (1 to r) processes (second round)
   For (i =1; i<= Q.length; i++);
      If (Q != NULL) then
         Schedule (P_i, RBT_i);
       Else
         Go to step 5
5. calculate AWT, ATAT,ART and CS
  //AWT=Average waiting time
  //ATAT=Average turnaround time
  //CS= Number of context switch
  //ART =Average Response Time
6. End

Figure 3.1: MMRRA ALGORITHM

3.3 MMRRA FLOW CHART

Below is the Flow Chart for the proposed work, the Modified Median Round Robin (MMRRA)

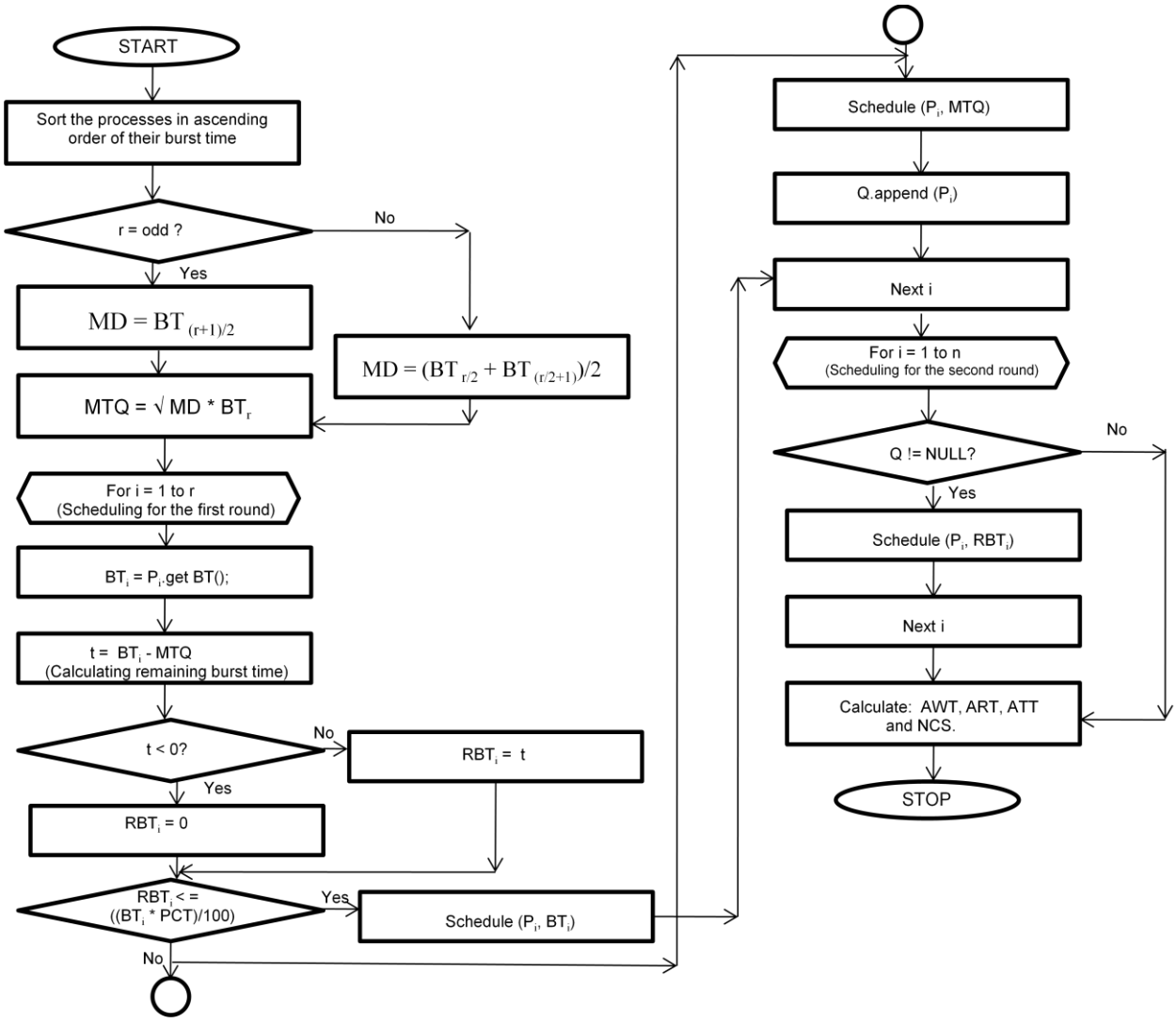

Figure 3.2: MMRRA FLOWCHART

3.4 EVALUATION: CLASSICALRR, IRR, IMRR, HLVQTRR, DRRCP & MMRRA

3.4.1 Generated data from the proposed algorithm

Table 3.1 shows five processes and their burst time selected at random. These values are to be used to evaluate and compare the RR CPU scheduling algorithms using analytic technique. As earlier mentioned, all processes are assumed to be in the ready queue and their arrival time is set to be zero.

Table 3.1: Processes' CPU burst time

PROCESSES	CPU BURST TIME
P0	71
P1	17
P2	70
P3	74
P4	66

3.4.1.1 Classical Round Robin

P0=71, P1=17, P2=70, P3=74, P4=66

In a classical RR the quantum time is the average of processes CPU burst time in the ready queue. Therefore, Quantum time (QT) = (71+17+70+74+66)/5 = 298/5 = 60

Remaining CPU burst after round 1

P0=11, P2= 10, P3=14, P4=6

The Gantt chart (Fig 3.3) is as follows:

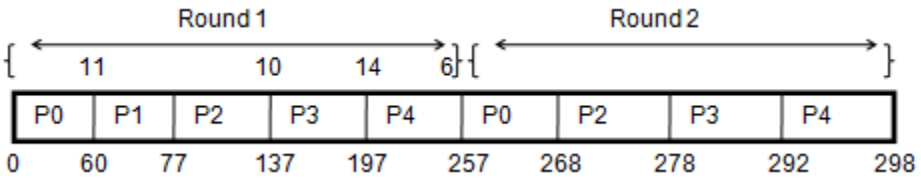

Figure 3.3: Classical RR Gantt Chart (1)

Table 3.2 displayed the average waiting time, average turnaround time, number of context switching and average response time for classical RR.

Table 3.2: Classical RR analytic Result (1)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P0: 257- 60 = 197 P1: 60 P2: 77+ 131 = 208 P3: 137 + 81 = 218 P4: 197 + 35 = 232	P0: 197 + 71 = 268 P1: 60 + 17 = 77 P2: 208 + 70 = 278 P3: 218 + 74 = 292 P4: 232 + 66 = 298	P0: 0 P1: 60 P2: 77 P3: 137 P4: 197
AWT= 183	**ATAT= 242.6**	**NCS = 4** **ART= 94.2**

3.4.1.2 *Improved Round Robin(IRR)*

As we said earlier, IRR picks the first process from the ready queue and allocate the CPU to it for a time interval of up to 1 time quantum. After completion of process's time quantum, it checks the remaining CPU burst time of the currently running process. If the remaining CPU burst time of the currently running process is less than 1 time quantum, the CPU again allocated

to the currently running process for remaining CPU burst time. In this case this process will finish execution and it will be removed from the ready queue. The scheduler then proceeds to the next process in the ready queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than 1 time quantum, the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

Method

{P0=71, P1=17, P2=70, P3=74, P4=66}

TQ = The average of the burst time of the processes = 60

The Gantt chart Figure 3.4 is as follows:

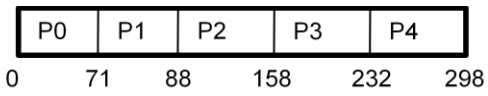

Figure 3.4: IRR Gantt chart (1)

Table 3.3 displayed the average waiting time, average turnaround time, number of context switching and average response time for IRR.

Table 3. 3: IRR analytic result (1)

Waiting Time	Turnaround Time TAT=WT+CPU burst	Response Time In first round only
P0: = 0 P1: = 71 P2: = 88 P3: = 158 P4: = 232	P0: 0 + 71 = 71 P1: 71 + 17 = 88 P2: 88 + 70 = 158 P3: 158 + 74 = 232 P4: 232 + 66 = 298	P0: = 0 P1: = 71 P2: = 88 P3: = 158 P4: = 232
AWT= 109.8	**ATAT= 169.4**	**NCS = 0** **ART=109.8**

3.4.1.3 Improved Mean Round Robin with Shortest Job First (IMRRSJF)

This algorithm used Round Robin with shortest Job first scheduling. The TQ studied to improve the efficiency of RR and performs degrades with respect to context switching, Average Wait Time and Average turnaround time. This approach was proposed to calculate the TQ, as the square root of mean times the Highest Burst.

Method:

P0=71, P1=17, P2=70, P3=74, P4=66

The sorted processes in ascending order are as follows:.

Processes	P1	P4	P2	P0	P3
CPU Burst Time	17	66	70	71	74

Quantum time (QT) = √ Mean* Highest burst

Mean=60

QT=√ 60*74 =√4440 =67

Remaining CPU burst after round 1

P2=3, P0=4, P3=7

The Gantt chart (Figure 3.5) is as follows:

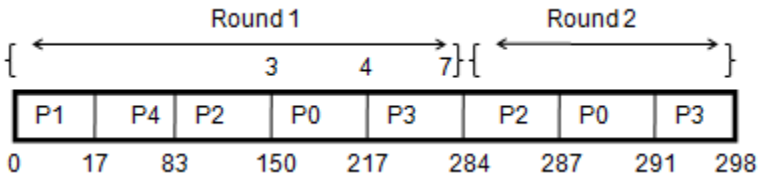

Figure 3.5: IMRRSJF Gantt chart (1)

Table 3.4 displayed the average waiting time, average turnaround time, number of context switching and average response time for IMRRSJF.

Table 3.4: IMRRSJF analytic result (1)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P0: = 150 + 70 = 220 P1: = 0 P2: = 83 + 134 = 217 P3: = 217 + 7 = 224 P4: = 17	P0: 220 + 71 = 291 P1: 0 + 17 = 17 P2: 217 + 70 = 287 P3: 224 + 74 = 298 P4: 17 + 66 = 83	P0: =15 0 P1: = 0 P2: = 83 P3: = 217 P4: = 17
AWT= 135.6	**ATAT= 195.2**	**NCS = 3** **ART= 93.4**

3.4.1.4 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 round, the remaining half should run to completion. This will greatly improve multiprogramming irrespective in the variation of processes' length in the ready queue. In SJF, the major problem is starvation. This is solved because only half of a process's job is executed what is left is preempted and the next process is attended to and so on.

Method:

Using the same dataset: P0=71, P1=17, P2=70, P3=74, P4=66 to evaluate HLVQTRR. Table 3.5 shows how the quantum time is calculated. In the first round, the quantum time is half each process burst time while in the second round; each process will run to completion. That is QT will be equal to each processes' remaining burst at second round.

Table 3.5: Calculation of QT for HLVQTRR (1)

Round 1 Quantum time	Remaining processes CPU burst after round 1	Round 2 Quantum time
P0: quantum time = 71/2 = 36	P0 = 35	P0: quantum time = 35
P1: quantum time = 17/2 = 9	P1 =8	P1: quantum time = 8
P2: quantum time = 70/2 = 35	P2 = 35	P2: quantum time = 35
P3: quantum time = 74/2 = 37	P3 = 37	P3: quantum time = 37
P4: quantum time = 66/2 =33	P4 = 33	P4: quantum time = 33

The Gantt chart (Figure 3.6) is as follows:

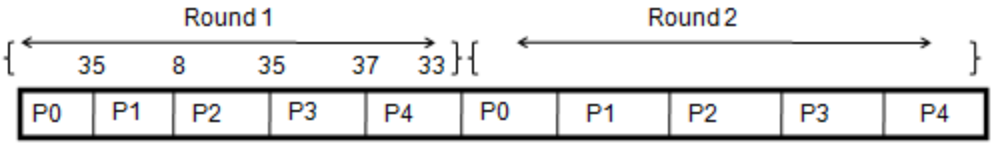

Figure 3.6: HLVQTRR Gantt chart (1)

Table 3.6: displayed the average waiting time, average turnaround time, number of context switching and average response time for HLVQTRR.

Table 3.6: HLVQTRR analytic result (1)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P0: = 114 P1: = 36 + 140 = 176 P2: = 45 + 113 = 158 P3: = 80 + 111 = 191 P4: = 117 + 115 =232	P0: 114 + 71 = 185 P1: 176 + 17 = 193 P2: 158 + 70 = 228 P3: 191 + 74 = 265 P4: 232 + 66 = 298	P0: = 0 P1: = 36 P2: = 45 P3: = 80 P4: = 117
AWT= 174.2	**ATAT= 233.8**	**NCS = 5**	**ART= 55.6**

3.4.1.5 Dynamic Round Robin with Controlled Preemption (DRRCP)

The main concern of this technique is the control applied on preemption to processes that are using the CPU in RR scheduling. The classical RR uses the average of processes in the ready queue as the quantum time, so is DRRCP. The proposal is if the quantum time allows a process to execute up to 95% of its job, it should be allowed to execute its left over job without being preempted. In this case, a process using the CPU may or may not be preempted even if its

quantum time is exhausted. It can only be preempted if its quantum time finishes and within its time slice it only processed less than 95% of its job. On the other hand, if the quantum time finishes and greater than or equal to 95% of the job is processed it should be allowed to run to completion, otherwise it should be preempted.

Method

Using the same dataset: P0=71, P1=17, P2=70, P3=74, P4=66 to evaluate DRRCP.

Quantum time (QT) = Average = 60

P0: 71 * 95/100 = 68, P1: 17 * 95/100 = 16
P2: 70 * 95/100 = 67 P3: 74 * 95/100 = 70
P4: 66 * 95/100 = 63

If 95% of the process CPU burst is less or equal to the quantum time, it should run to completion, otherwise it should be preempted if its time slice is expired. The Gantt chart (Figure 3.7) is as follows:

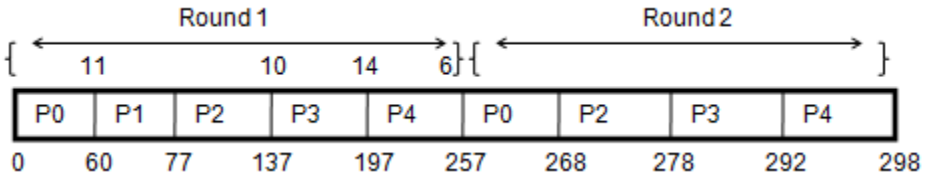

Figure 3.7: DRRCP Gantt chart (1)

Table 3.7 displayed the average waiting time, average turnaround time and the Number of context switching for DRRCP

Table 3.7: DRRCP analytic result (1)

Waiting Time	Turnaround Time TAT=WT+CPU burst	Response Time In first round only
P0: = 197 P1: = 60 P2: = 77 + 131 = 208 P3: = 137 + 81 = 218 P4: = 197 + 35 = 232	P0: 197 + 71 = 268 P1: 60 + 17 = 77 P2: 208 + 70 = 278 P3: 218 + 74 = 292 P4: 232 + 66 = 298	P0: = 0 P1: = 60 P2: = 77 P3: = 137 P4: = 197
AWT= 183	**ATAT= 242.6**	**NCS = 4**	**ART= 94.2**

3.4.1.6 Modified Median Round Robin Algorithm (MMRRA)

CPU scheduling is the basis of multiprogrammed Operating System. Most CPU scheduling algorithms Concentrate on maximizing throughput, CPU utilization and minimizing waiting time, response

Time, turnaround time and number of context switching for a set of requests. Many algorithms have been proposed which are easy to implement but generally most of these algorithms does not provide the best service. MMRRA describes an improvement in RR.

The TQ is Generated to improve the efficiency of RR. There by, producing a reasonable: context Switching, Average waiting time, Average turnaround time and Average response time.

Method:

-The processes are arranged in ascending order of their burst time

-The time quantum will be calculated as: TQ=√Median *HB.

-The remaining burst of each process will be calculated.

-The shortest job will be assigned to the CPU using the time quantum generated.

-If the remaining burst is greater than 20% of the process's burst then it will join the queue at the tail else the process will be allow to run to completion and leave the queue.

Using the same dataset: P0 =71, P1=17, P2=70, P3=74, P4= 66 to evaluate MMRRA, Sort the burst time in ascending order:

The Sorted processes in ascending order is as follows:

Processes	P1	P4	P2	P0	P3
CPU burst time	17	66	70	71	74

Median = 70

QT = √Median*HB = √(70 *74) = 72.

Table 3.8 displayed the Status of each process.

Table 3.8: The status of each process (MMRRA 1)

Process burst time	Analysis of the Remaining burst time of the processes	Status
P1 = 17	P1 = 17 – 72 < (20% of P1)	Run to completion
P4 = 66	P4 = 66 – 72 < (20% of P4)	Run to completion
P2 = 70	P2 = 70 – 72 < (20% of P2)	Run to completion
P0 = 71	P0 = 71 – 72 < (20% of P0)	Run to completion
P3 = 74	P3 = 74 – 72 < (20% of P3)	Run to completion

The Gantt chart (Figure 3,8) is as follows:

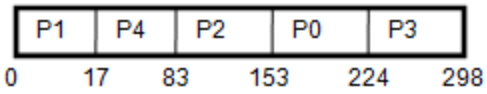

Figure 3.8: MMRRA Gantt chart (1)

Table 3.9 displayed the average waiting time, average turnaround time average response time and the number of context switching for MMRRA.

Table 3.9: MMRRA analytic result (1)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P0: = 153 P1: = 0 P2: = 83 P3: = 224 P4: = 17	P0 = 153 + 71 = 224 P1 = 0 + 17 = 17 P2 = 83 + 70 = 153 P3 = 224 + 74 = 298 P4 = 17 + 66 = 83	P0: = 153 P1: = 0 P2: = 83 P3: = 224 P4: = 17
AWT= 95.4	**ATAT= 155**	**NCS = 0**	**ART= 95.4**

Table 3.10 displayed the summary results of the six algorithms looking at their average waiting time, average turnaround time, number of context switching and average response time.

Table 3.10: Evaluation summary of the analytic result (1)

Criteria	**CLASSICAL RR**	**IRR**	**IMRRSJF**	**HLVQTRR**	**DRRCP**	**MMRRA**
AWT	183	109.8	135.6	174.2	183	95.4
ATAT	242.6	169.4	195.2	233.8	242.6	155
ART	94.2	109.8	93.4	55.6	94.2	95.4
NCS	4	0	3	5	4	0

From Table 3.10, it can be observed that the **AWT** and **ATT** of MMRRA is far more better than the other five algorithms then followed by IRR. IMRRSJF is better than only DRRCP HLVQTRR and CLASSICAL RR. Next is the HLVQTRR. The CLASSICAL RR and the DRRCP have the same result and it is the worst.

But in terms of **ART** HLVQTRR has the best result followed by IMRRSJF that has 93.4ms next are the DRRCP and CLASSICAL RR with the same result of 94.2ms. MMRRA followed with 95.4ms. If you look at the result of IMRRSJF, CLASSICAL RR and DRRCP, the difference with that of MMRRA is not more than 2ms which is insignificant. IRR has the worst result.

In terms of **NCS**, MMRRA and IRR are the best then followed by IMRRSJF. DRRCP and CLASSICAL RR have the same result that is better than HLVQTRR(meaning, HLVQTRR has the worst result). It is clearly shown that the result of CLASSICAL RR and DRRCP are exactly the same. The reason behind this is that, for DRRCP there is no small left over for a process burst that exceed the QT which can be handled by the DRRCP. That is the reason why everything works like Classical RR.

If we look at the overall result, we can observed that MMRRA has the best result on three criteria that is AWT, ATAT and NCS. In terms of ART, HLVQTRR is the best. Though, IMRRSJF,CLASSICAL RR and DRRCP were better than MMRRA but still their difference is not more than 2ms which is insignificant.

Conclusively, this shows that ***MMRRA*** is better than all the five scheduling algorithms.

3.4.2 Generated data from DRRCP

Table 3.11 Illustrate DRRCP processes and their CPU burst time. These values are to be used to evaluate and compare the RR CPU scheduling algorithms using analytic technique. As earlier mention, all processes are assumed to be in the ready queue and their arrival time is set to be zero.

Table 3 11: DRRCP processes' CPU burst time

PROCESSES	CPU BURST TIME
P1	110
P2	89
P3	113
P4	137
P5	86
P6	131
P7	95

3.4.2.1 Classical Round Robin

P1=110, P2=89, P3=113, P4=137, P5=86, P6=131, P7=95

In a classical RR the quantum time is the average of processes CPU burst time in the ready queue. Therefore, Quantum time (QT) = (110+89+113+137+86+131+95)/7 =109

The Gantt chart (Figure 3.9) is as follows:

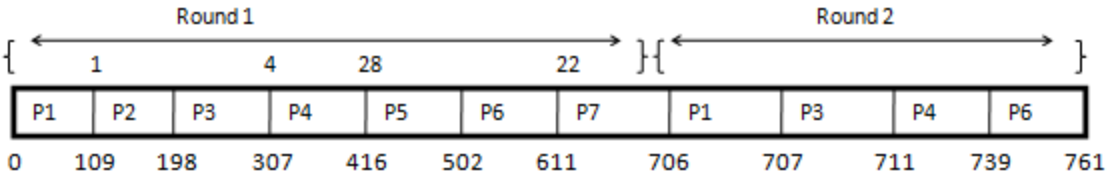

Figure 3.9: Classical Round Robin Gantt chart (2)

Table 3.12 displayed the average waiting time, average turnaround time, number of context switching and average response time for classical RR.

Table 3. 12: Classical RR analytic result (2)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 706 - 109 = 597 P2: 109 P3: 198 + 400 = 598 P4: 307 + 295 = 602 P5: 416 P6: 502 + 128 = 630 P7: 611	P1: 597 + 110 = 707 P2: 109 + 89 = 198 P3: 598 + 113 = 711 P4: 602 + 137 = 739 P5: 416 + 86 = 502 P6: 630 + 131 = 761 P7: 611 + 95 = 706	P1: 0 P2:109 P3: 198 P4: 307 P5: 416 P6: 502 P7: 611
AWT= 509	**ATAT= 617.7**	**NCS = 4** **ART= 306.1**

3.4.2.2 Improved Round Robin (IRR)

TQ = The average of the burst time of the processes.

TQ= (110+89+113+137+86+131+95)/7 =109

If the remaining burst time< TQ it will run to completion.

The Gantt chart (Figure 3.10) is as follows:

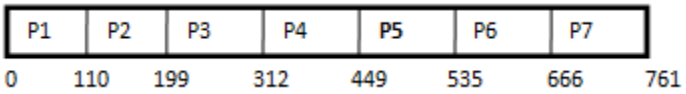

Figure 3.10: IRR Gant Chart (2)

Table 3.13 displayed the average waiting time, average turnaround time, number of context switching and average response time for IRR.

Table 3.13: IRR analytic result (2)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 0 P2: 110 P3: 199 P4: 312 P5: 449 P6: 535 P7: 666	P1: 0 + 110 = 110 P2: 110 + 89 = 199 P3: 199 + 113 = 312 P4: 312 + 137 = 449 P5: 449 + 86 = 535 P6: 535 + 131 = 666 P7: 666 + 95 = 761	P1: 0 P2:110 P3: 199 P4: 312 P5: 449 P6: 535 P7: 666
AWT= 324	**ATAT= 433**	**NCS = 0** **ART= 324**

3.4.2.3 Improved Mean Round Robin with Shortest Job First (IMRRSJF)

Below are the sorted processes in ascending order

Processes	P5	P2	P7	P1	P3	P6	P4
Burst Time	86	89	95	110	113	131	137

Mean =109, Highest burst time (HB) =137.

QT = √(mean * HB) = √(109*137) = √14933 = 122

The Gantt chart (Figure 3.11) is as follows:

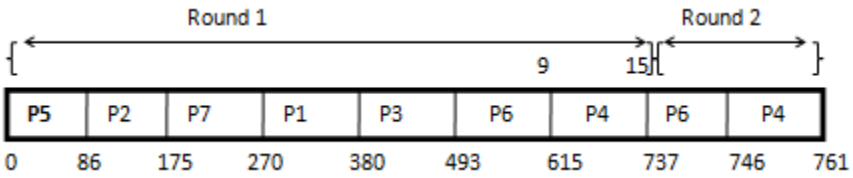

Figure 3.11: IMRRSJF chart (2)

Table 3.14 displayed the average waiting time, average turnaround time, number of context switching and average response time for IMRRSJF.

Table 3.14: IMRRSJF analytic result (2)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 270 P2: 86 P3: 380 P4: 615 + 9 = 624 P5: 0 P6: 493 + 122 = 615 P7: 175	P1: 270 + 110 = 380 P2: 86 + 89 = 175 P3: 380 + 113 = 493 P4: 624 + 137 = 761 P5: 0 + 86 = 86 P6: 615 + 131 = 746 P7: 175 + 95 = 270	P1: 270 P2:86 P3: 380 P4: 615 P5: 0 P6: 493 P7: 175
AWT= 307	**ATAT= 416**	**NCS = 2** **ART= 288**

3.4.2.4 Half Life Variable Quantum Time Round Robin (HLVQTRR)

Method:

Table 3.15 shows how the quantum time is calculated. In the first round, the quantum time is half each process burst time while in the second round; each process will run to completion. That is QT will be equal to each processes' remaining burst at second round.

Table 3.15: Calculation of QT for HLVQTRR (2)

Round 1 Quantum time	Remaining processes CPU burst after round 1	Round 2 Quantum time
P1: quantum time = 110/2 = 55	P1 = 55	P1: quantum time = 55
P2: quantum time = 89/2 = 45	P2 =44	P2: quantum time = 44
P3: quantum time = 113/2 = 57	P3 = 56	P3: quantum time = 56
P4: quantum time = 137/2 = 69	P4 = 68	P4: quantum time = 68
P5: quantum time = 86/2 = 43	P5 = 43	P5: quantum time = 43
P6: quantum time = 131/2 =66	P6 = 65	P6: quantum time = 65
P7: quantum time = 95/2 =48	P7 = 47	P7: quantum time = 47

The Gantt chart (Figure 3.12) is as follows:

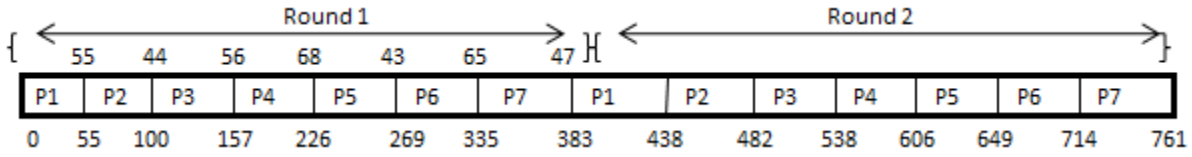

Figure 3.12: HLVQTRR Gantt chart (2)

Table 3.16: displayed the average waiting time, average turnaround time, number of context switching and average response time for HLVQTRR.

Table 3.16: HLVQTRR analytic result (2)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 383 + 55 = 328 P2: 55 + 338 = 393 P3: 100 + 325 = 425 P4: 157 + 312 = 469 P5: 226 + 337 = 563 P6: 269 + 314 = 583 P7: 335 + 331 = 666	P1: 328 + 110 = 438 P2: 393 + 89 = 482 P3: 425 + 113 = 538 P4: 469 + 137 = 606 P5: 563 + 86 = 649 P6: 583 + 131 = 714 P7: 666 + 95 = 761	P1: 0 P2: 55 P3: 100 P4: 157 P5: 226 P6: 269 P7: 335
AWT= 490	**ATAT= 598**	**NCS = 7** **ART= 163**

3.4.2.5 Dynamic Round Robin with Controlled Preemption (DRRCP)

Method:

QT = Mean = 109

NINETY FIVE PERCENT (95%)

P1: 110*95/100 ≈104.5 P2:89*95/100 ≈ 84.6 P3: 113*95/100 ≈ 107.4

P4: 137*95/100 ≈ 130.1 P5:86*95/100 ≈ 81.7 P6: 131*95/100 ≈124.5

P7: 95 *95/100 ≈ 90.3

If 95% of the process CPU burst is less or equal to the quantum time, it should run to completion, otherwise it should be preempted if its time slice is expired.

The Gantt chart (Figure 3.13) is as follows:

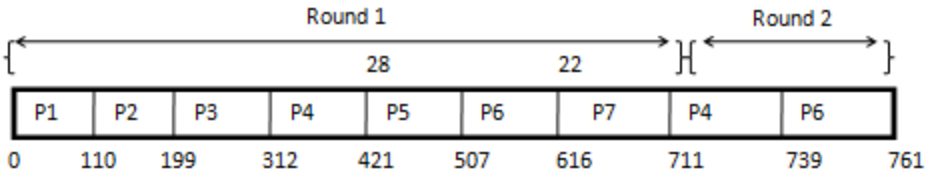

Figure 3.13: DRRCP Gantt chart (2)

Table 3.17 displayed the average waiting time, average turnaround time and the Number of context switching for DRRCP.

Table 3.17: DRRCP analytic result (2)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 0 P2: 110 P3: 199 P4: 312 + 290 = 602 P5: 421 P6: 507 + 123 = 630 P7: 616	P1: 0 + 110 = 110 P2: 110 + 89 = 199 P3: 199 + 113 = 312 P4: 602 + 137 = 739 P5: 421 + 86 = 507 P6: 630 + 131 = 761 P7: 616 + 95 = 711	P1: 0 P2:110 P3: 199 P4: 312 P5: 421 P6: 507 P7: 616
AWT= 368	**ATAT= 477**	**NCS = 2** **ART= 309**

3.4.2.6 Modified Median Round Robin Algorithm (MMRRA)

Method: Using the same dataset: the processes are sorted in ascending order of their burst time:

P5= 86, P2=89, P7=95, P1=110, P3=113, P6 =131, P4 =137

Median = 110

QT = √(Median*HB) = √(110 *137) = √15070 = 123.

Table 3.18 displayed the remaining burst time of each process.

Table 3.18: The status of each process (MMRRA 2)

Process burst time	Analysis of the Remaining burst time of the processes	Status
P5 = 86	86 – 123 < (20% of P5)	Run to completion
P2 = 89	89 – 123 < (20% of P2)	Run to completion
P7 = 95	95 – 123 < (20% of P7)	Run to completion
P1 = 110	110 - 123 < (20% of P1)	Run to completion
P3 = 113	113 - 123 < (20% of P3)	Run to completion
P6 = 131	131 - 123 < (20% of P6)	Run to completion
P4 = 137	137 – 123 < (20% of P4)	Run to completion

The Gantt chart (Figure 3.14) is as follows:

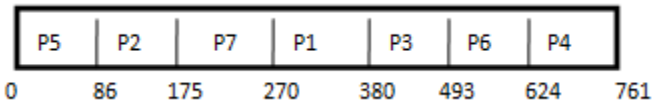

Figure 3.14: MMRRA Gantt chart (2)

Table 3.19 displayed the average waiting time, average turnaround time, response time and the number of context switching for MMRRA

Table 3.19: MMRRA analytic result (2)

Waiting Time	Turnaround Time TAT=WT+CPU burst	Response Time In first round only
P1: 270 P2: 86 P3: 380 P4: 624 P5: 0 P6: 493 P7: 175	P1: 270 + 110 = 380 P2: 86 + 89 = 175 P3: 380 + 113 = 493 P4: 624 + 137 = 761 P5: 0 + 86 = 86 P6: 493 + 131 = 624 P7: 175 + 95 = 270	P1: 270 P2:86 P3: 380 P4: 624 P5: 0 P6: 493 P7: 175
AWT= 290	**ATAT= 398**	**NCS = 0** **ART= 290**

Table 3.20 displayed the summary results of the six algorithms looking at their average waiting time, average turnaround time, number of context switching and average response time.

Table 3.20: Evaluation summary of the analytic result (2)

Criteria	CLASSICAL RR	IRR	IMRRSJF	HLVQTRR	DRRCP	MMRRA
AWT	509	324	307	489.5	368	290
ATAT	618	433	416	598	477	398
ART	306	324	288	163	309	290
NCS	4	0	2	7	2	0

From Table 3.20, it can be observed that, the **AWT** and **ATT** of MMRRA is far more better than the other five algorithms then followed by IMRRSJF. Next is the IRR for the remaining three, DRRCP is better than HLVQTRR and it is clearly shown that Classical has the worst result.

But in terms of **ART** HLVQTRR has the best result followed by IMRRSJF that has 288ms next is the MMRRA with 290ms and its difference with the IMRRSJF is just 2ms. Next is the Classical then followed by DRRCP. IRR has the worst result as in the first case. If you look at the result of DRRCP and Classical RR, DRRCP performs better on like in the 1st case where they have the same result. The reason behind this is that DRRCP took care of 2 processes P1 and P3 that have left over of 1 and 4 respectively(meaning, they run to completion without going for second round).

In terms of **NCS**, IRR and MMRRA has the best result followed by IMRRSJF and DRRCP that have the same result. CLASSICAL is better than HLVQTRR meaning that HLVQTRR has the worst result in terms of CS.

If we look at the overall result, we can observed that MMRRA has the best result in three criteria that is, AWT, ATAT and NCS. In terms of ART, HLVQTRR is the best where IMRRSJF followed then the MMRRA. But it is clearly shown that the difference between the MMRRA and IMRRSJF is just that of 2ms which is insignificant.

Conclusively, this shows that ***MMRRA*** is better than all the five scheduling algorithms}.

3.4.3 Generated dada from IMRRSJF

Table 3.21 Illustrate IMRRSJF processes and their CPU burst time. These data set are to be used to evaluate and compare the RR CPU scheduling algorithms using analytic technique.

As earlier mention, processes are assumed to be in the ready queue and their arrival time is set to be zero.

Table 3.21: IMRRSJF Processes' CPU burst time

PROCESSES	CPU BURST TIME
P1	80
P2	52
P3	35
P4	22
P5	11

3.4.3.1 Classical Round Robin

P1=80, P2=52, P3=35, P4=22, P5=11

(QT) = Average = 40

The Gantt chart (Figure 3.15) is as follows:

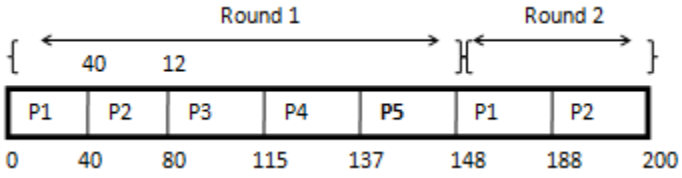

Figure 3.15: Classical RR Gantt Chart (3)

Table 3.22 displayed the average waiting time, average turnaround time, number of context switching and average response time for classical RR.

Table 3.22: Classical RR analytical result

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 148- 40 = 108 P2: 40 + 108 = 148 P3: 80 P4: 115 P5: 137	P1: 108 + 80 = 188 P2: 148 + 52 = 200 P3: 80 + 35 = 115 P4: 115 + 22 = 137 P5: 137 + 11 = 148	P1: 0 P2: 40 P3: 80 P4: 115 P5: 137
AWT= 118	**ATAT= 158**	**NCS = 2** **ART= 74**

3.4.3.2 Improved Round Robin(IRR)

TQ = The average of the burst time of the processes.

TQ= 40

If the remaining burst time < TQ it will run to completion.

The Gantt chart (Figure 3.16) is as follows:

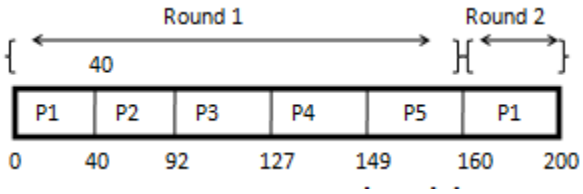

Figure 3.16: IRR Gant chart (3)

Table 3.23 displayed the average waiting time, average turnaround time, number of context switching and average response time for IRR.

Table 3.23: IRR analytical result (3)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 160-40=120 P2: 40 P3: 92 P4: 127 P5: 149	P1: 120+80=200 P2: 40+52=92 P3: 92+35=127 P4: 127+22=149 P5: 149+11=160	P1: 0 P2: 40 P3: 92 P4: 127 P5: 149
AWT= 106	**ATAT= 146**	**NCS = 1** **ART= 82**

3.4.3.3 Improved Mean Round Robin With Shortest Job First (IMRRSJF)

The sorted processes in ascending order:

Processes	P5	P4	P3	P2	P1
Burst Time	11	22	35	52	80

Mean =40, Highest burst time (HB) = 80.

QT = √(mean * HB) = √(40*80) = √3200 = 56

The Gantt chart (Figure 3.17) is as follows:

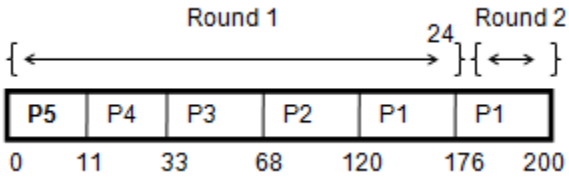

Figure 3.17: IMRRSJF Gantt chart (3)

Table 3.24 displayed the average waiting time, average turnaround time, number of context switching and average response time for IMRRSJF.

Table 3.24: IMRRSJF analytic result (3)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 120 P2: 68 P3: 33 P4: 11 P5: 0	P1: 120+80=200 P2: 68+52=120 P3: 33+35=68 P4: 11+22=33 P5: 0+11=11	P1: 120 P2: 68 P3: 33 P4: 11 P5: 0
AWT= 46	**ATAT= 86**	**NCS = 1**	**ART= 46**

3.4.3.4 Half Life Variable Quantum Time Round Robin (HLVQTRR)

Method:

Table 3.25 shows how the quantum time is calculated.

Table 3. 25: Calculation of QT for HLVQTRR (3)

Round 1 Quantum time	Remaining processes CPU burst after round 1	Round 2 Quantum time
P1: quantum time = 80/2 = 40	P1 = 40	P1: quantum time = 40
P2: quantum time = 52/2 = 26	P2 =26	P2: quantum time = 26
P3: quantum time = 35/2 = 18	P3 = 17	P3: quantum time = 17
P4: quantum time = 22/2 = 11	P4 = 11	P4: quantum time = 11
P5: quantum time = 11/2 =6	P5 = 5	P5: quantum time = 5

The Gantt chart (Figure 3.18) is as follows:

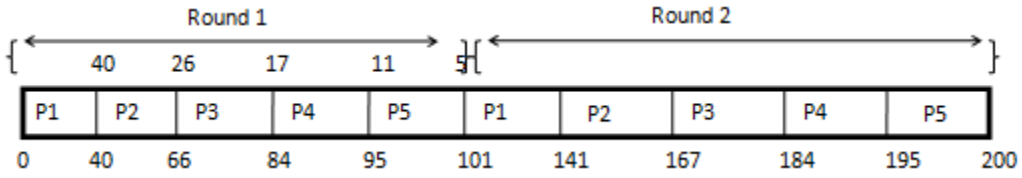

Figure 3.18: HLVQTRR Gantt Chart (3)

Table 3.26 displayed the average waiting time, average turnaround time, number of context switching and average response time for IMRRSJF.

Table 3.26: HLVQTRR analytic result (3)

Waiting Time	Turnaround Time TAT=WT+CPU burst	Response Time In first round only
P1: 61 P2: 115 P3: 149 P4: 173 P5: 189	P1: 61+80=141 P2: 115+52=167 P3: 149+35=184 P4: 173+22=195 P5: 189+11=200	P1: 0 P2: 40 P3: 66 P4: 84 P5: 95
AWT= 137.4	**ATAT= 177.4**	**NCS = 5** **ART= 57**

3.4.3.5 Dynamic Round Robin with Controlled Preemption (DRRCP)

Method: QT = Mean = 40

NINETY FIVE PERCENT (95%) OF THE PROCESS BURST TIME

P1: 80*95/100 ≈ 76 P2: 52*95/100 ≈ 49 P3: 35*95/100 ≈ 33

P4: 22*95/100 ≈ 21 P5:11*95/100 ≈ 11

If 95% of the process CPU burst is less or equal to the quantum time, it should run to completion, otherwise it should be preempted if its time slice is expired.

The Gantt chart (Figure 3.19) is as follows:

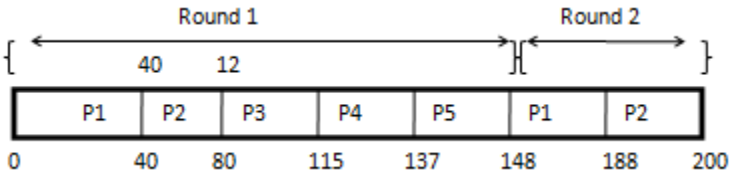

Figure 3.19: DRRCP Gantt Chart (3)

Table 3.27 displayed the average waiting time, average turnaround time, number of context switching and average response time for IMRRSJF.

Table 3.27: DRRCP analytic result (3)

Waiting Time	Turnaround Time TAT=WT+CPU burst	Response Time In first round only
P1: 108 P2: 148 P3: 80 P4: 115 P5: 137	P1: 108 + 80 = 188 P2: 148 + 52 = 200 P3: 80 + 35 = 115 P4: 115 + 22 = 137 P5: 137 + 11 = 148	P1: 0 P2: 40 P3: 80 P4: 115 P5: 137
AWT= 117.6	**ATAT= 157.6**	**NCS = 2** **ART= 74.4**

3.4.3.6 Modified Median Round Robin Algorithm (MMRRA)

Method: Using the same dataset: the processes are sorted in ascending order of their burst time:

P5= 11, P4=22, P3=35, P2=52, P1=80

Median = 35

QT = √(Median*HB) = √(35 *80) = √2800 = 53.

Table 3.28 displayed the remaining burst time of each process.

Table 3.28: The status of each processes (MMRRA 3)

Process burst time	Analysis of the Remaining burst time of the processes	Status
P5 = 11	P5 = 11 – 53 < (20% of P5)	Run to completion
P4 = 22	P4 = 22 – 53 < (20% of P4)	Run to completion
P3 = 35	P3 = 35 – 53 < (20% of P3)	Run to completion
P2 = 52	P2 = 52 – 53 < (20% of P2)	Run to completion
P1 = 80	P1 = 80 – 53 > (20% of P1)	Go for second round

The Gantt chart (Figure 3.20) is as follows:

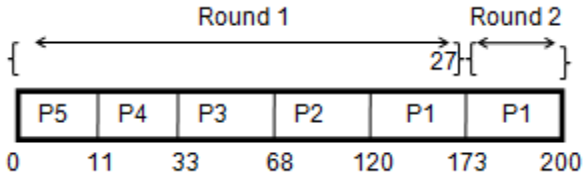

Figure 3.20: MMRRA Gantt Chart (3)

Table 3.29 displayed the average waiting time, average turnaround time, response time and the number of context switching for MMRRA.

Table 3.29: MMRRA analytic result (3)

Waiting Time	**Turnaround Time** TAT=WT+CPU burst	**Response Time** In first round only
P1: 120 P2: 68 P3: 33 P4: 11 P5: 0	P1: 120+80=200 P2: 68+52=120 P3: 33+35=68 P4: 11+22=33 P5: 0+11=11	P1: 120 P2: 68 P3: 33 P4: 11 P5: 0
AWT= 46.4	**ATAT= 86.4**	**NCS =1** **ART= 46.4**

Table 3.30 displayed the summary results of the six algorithms looking at their average waiting time, average turnaround time, number of context switching and average response time.

Table 3.30: Evaluation summary of the analytic result (3)

Criteria	CLASSICAL RR	IRR	IMRRSJF	HLVQTRR	DRRCP	MMRRA
AWT	117.6	105.6	46.4	137.4	117.6	46.4
ATAT	157.6	145.6	86.4	177.4	157.6	86.4
ART	74.4	81.6	46.4	57	74.4	46.4
NCS	2	1	1	5	2	1

From Table 3.30, it can be observed that: the **AWT** and **ATT** of MMRRA and IMRRSJF are the same and far more better than the other four algorithms then followed by IRR. Next are the DRRCP and Classical that have the same result just like for the 1st case. HLVQTRR this time around has the worst result.

But in terms of **ART,** MMRRA and IMRRSJF have the same and the best result(they are now ahead of HLVQTRR that use to have the best result on ART). Next is the HLVQTRR. DRRCP and Classical have the same result and are ahead of IRR.(this shows that IRR has the worst result as in the first and second cases).

In terms of **NCS**, MMRRA, IRR and IMRRSJF have the best result then followed by DRRCP and Classical that have the same result. HLVQTRR has the worst result as usual.

If we look at the overall result, we can observed that MMRRA and IMRRSJF have the best result in four criteria that is, ART, AWT, NCS and ATAT.

Conclusively, this shows that ***MMRRA & IMRRSJF*** are better than all the remaining four scheduling algorithms.

CHAPTER FOUR: EVALUATION OF THE DESIGNED MMRRA

4.1 INTRODUCTION

In chapter three, the six algorithms were analyzed using analytic technique with five and seven processes. But in order to understand the behavior of these algorithms, and draw valid conclusions, there is the need to develop a simulator that will consider wide range of dataset. This will be run for ny times for analysis. Using exponential distribution function, a simulator is designed to generate random variables which are to be used as processes and their associated CPU burst. This will be use to find their average waiting time, average turnaround time, number of context switching and average response time. Their results shall be presented in a tabula and graphically form for proper analysis.

4.2 DESCRIPTION OF THE SIMULATION

Simulations: This is the imitation of the operation of a real-world process or system over time. Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system (Bank et al., 2005). To get a more accurate evaluation of scheduling algorithms, we can use simulations (Silberschatz *et al.,* 2002).

Implementation: This approach puts the actual algorithm in the real system for evaluation under real operating conditions. The major difficult y with this approach is the high cost. The expense is incurred not only in coding the algorithm and modifying the operating system to support it

(along with its required data structures) but also in the reaction of the users to a constantly changing operating system.

The simulator was designed using Visual Basic 6.0 as the programming language and executed on windows 7 operating system. The size of the ram used is 2.0GB, the speed is 1.56GHz and the size of the hard disk is 300GB. The simulator considers all processes to be in the ready queue and their arrival time set to be zero. Also, processes with shortest burst are considered first . The simulator uses exponential distribution function which requires three input parameters to generate processes and their associated CPU burst. The input parameters are: mean (as lambda), number of process and percentage. These parameters are variables, meaning they can assume any value. The maximum number of process is set to be 9999, the maximum number of the mean (lambda) is set to be 999 and the percentage can assume any value from 1-100. When the program is running, it will request the user to supply number of process, the lambda and the percentage in that order. By running the simulator, it will automatically generate the processes and their associated CPU burst, and performed the Round Robin on those processes. Finally, it computes the average waiting time, average turnaround time, number of context switching, and average response time for each algorithm. All the findings gotten from the developed simulator shall be examine and discussed under some important criteria used in time sharing systems. Putting all features into consideration, the advantages and disadvantages shall be thoroughly discussed. These criteria are discussed below:

4.3 PSEUDOCODE FOR THE SIMULATION

Below is the Pseudo code used for the running of the Simulator.

- Initialise the Simulator
- Supply the parameter's value to the simulator

  //N = number of processes, Lambda & Percentage
- The result will be generated with 3 buttons.

  // The results are: AWT, ATAT, ART & NCS

  // The buttons are: <***Simulate***>, <***Graph***>& <***Simulate with different %***>
- Selecting <***Simulate***>, allow you to carry-out a different Simulation.
- Selecting <***Graph***>, show you the result for the current simulation graphically for each criteria.
- Selecting <***Simulate with different %***>, allow you to undergo an experiment

  for choosing the best % for the program.
- When carrying out the experiment, another button will appear: <***Compare Result***>.
- Selecting this button create a text file at the task bar.
- Selecting the file show the comparison of all the percentages you entered

  based on the four Criteria: AWT, ATAT, ART & NCS

Figure 4.1: Pseudo code for the Simulation

4.4 EVALUATION CRITERIA

Average waiting Time (AWT): A process should not wait for so long in the ready queue. If it happens, the process will appear to be blocked and this will affect the system performance. This is not good in real time systems. Waiting time is to be minimized.

Average Turnaround Time (ATAT): The turnaround time is a function of the waiting time. If the waiting time is high, the turnaround time will also be high. This criterion is to be minimized and the only way is by minimizing the waiting time.

Context switching (CS): The process of saving the content of a process and selecting another to run is called context switch. The state of one process is saved then preempted, while another is allocated the CPU. This may go on for some number of times depending chiefly on the quantum time or processes length. CS is an overhead because the CPU does nothing useful when context switching. This little time of CPU idleness is not to be welcomed because the CPU needs to be busy at all time and at all cost. A real time system cannot be useful without switching from one process to another. This is the main reason why Round Robin algorithm is introduced. So, RR is designed to perform switching, and as such consider good for real time systems. Since switching is necessary, it must be done in such a way to achieve optimal average waiting time and average turnaround time.

Response Time: The time a process starts receiving attention is called response time. When did a process start receiving feedback? In a real time system, waiting time is not enough to judge the performance of the system, there is a need to support the judgment with response time.

Multiprogramming: The degree of multiprogramming display by a system is an important factor in real time system. Some processes do not show any level of multiprogramming. That is they run to completion in the first round. It is clear that quantum time is a factor in achieving multiprogramming.

Fairness: All CPU scheduling algorithms try to be fair to all processes. An algorithm may be fair to one process but not to another. At least, FCFS is fair enough in the scene that a process must not be late, if it is late then it must follow queue. The process that arrived earlier will say it is fair, but those that are late with a negligible job size will not consider it to be fair at all. While SJF try to be fair to shorter processes but truly unfair to larger processes that may have arrived earlier. RR tries to fair by allocating time slice to processes to use the CPU and may be preempted for next process.

4.5 ALGORITHM EVALUATION

There are various evaluation methods for evaluating CPU Scheduling algorithms. This research work uses Simple Queuing Model.

Simple Queuing Model: A queuing system consists of one or more servers that provide service of some sort to arriving customers. Customers who arrive to find all servers busy generally join one or more queues (lines) in front of the servers, hence the name **queuing systems**. There are several everyday examples that can be described as queuing systems, such as bank-teller service, computer systems, manufacturing systems, maintenance systems, communications systems and so on.

Components of a Queuing System: A queuing system is characterised by three components:

Arrival process, Service mechanism and Queue discipline.

Arrival Process

Arrivals may originate from one or several sources referred to as the **calling population**. The calling population can be limited or 'unlimited'. An example of a limited calling population may

be that of a fixed number of machines that fail randomly. The arrival process consists of describing how customers arrive to the system.

Service Mechanism

The service mechanism of a queuing system is specified by the number of servers (denoted by s), Each server having its own queue or a common queue and the probability distribution of customer's service time.

Queue Discipline

Discipline of a queuing system means the rule that a server uses to choose the next customer from the queue (if any) when the server completes the service of the current customer. Commonly used queue disciplines are:

FIFO - Customers are served on a first-in first-out basis. LIFO - Customers are served in a last-in first-out manner. Priority - Customers are served in order of their importance on the basis of their service requirements. Round Robin- Customers are served base on quantum time allocated to them (Browne; 2003).

4.6 DISTRIBUTION FUNCTION USED

We have many distribution functions among which are the poison, normal, exponential etc. The Poisson distribution is applicable in situations where random "events" occur at a certain rate over a period of time. Poisson is a *Discrete* distribution: P(number of events) e.g. The hourly number of customers arriving at a bank, the hourly number of accesses to a particular web Server, the monthly number of employees who had an absence in a large company, monthly demands for a particular product etc. All of these are situations where the Poisson distribution may be applicable.

Exponential distribution is another useful continuous distribution. In this paper, we decided to use exponential distribution because It models time between independent randomly occurring events, where frequency of events is independent of time- P(time till first event). Eg. The length of time between telephone calls, the length of time between arrivals at a service station etc. this situation is very similar to our process scheduling since it considers inter-arrival times, service times etc that are assumed to be exponential distribution (OPRE; 6301).

4.7 ALGORITHM FOR THE EXPONENTIAL DISTRIBUTION

The algorithm that generated the random number using exponential distribution is shown below: (Abur *et al*., 2011)

```
1. Public Function rndom() As Integer
2. Dim Lambda As Integer = 1
3. Dim seed As Single = 0
4. Dim X As Integer = 0
5. Randomize()
6. seed = Rnd()
7. X = seed * Lambda
8. rndom = Int(1 - Exp(-Lambda * X) * 10)
9. Debug.Print(Exp(-Lambda * X))
10. Debug.Print(Exp(1))
11. End Function
```

4.8 THE BEST WAY OF CHOOSING PERCENTAGE (for a method)

From the observation made on most of the proposed work reviewed, they tempt to pick a

percentage (%) to be used in their method that they might feel it will solve their problem without finding a concrete reason for picking such percentage. For this research work, an experiment has been carried out using the simulator to find the best percentage to use, which we came up with 20% as the best. The following graphs shows the analysis:

The graph of Figure 4.2 and Figure 4.3 for AWT and ATAT respectively shows that 20% is the best percentage to use for this method. The graphs, show that the line for 20% is below 15, 10 and 5 percent. Meaning, the AWT and ATAT is better for the 20% than the 15, 10 and 5.

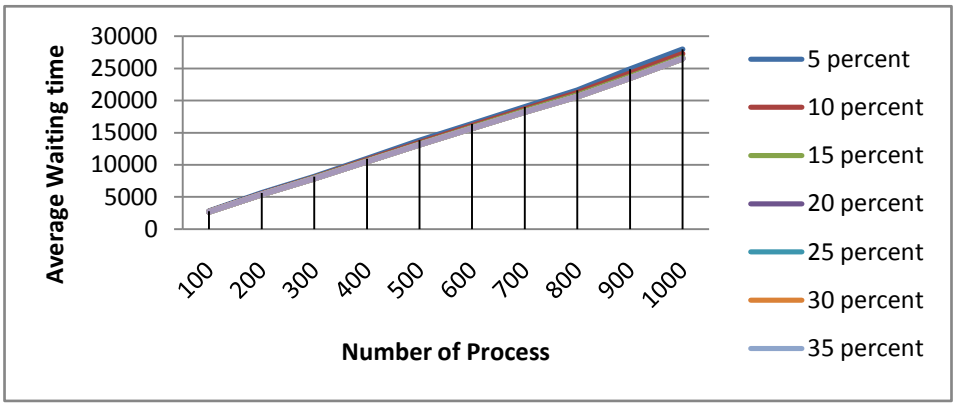

Figure 4.2: Percentage (%) comparison using Average waiting time graph

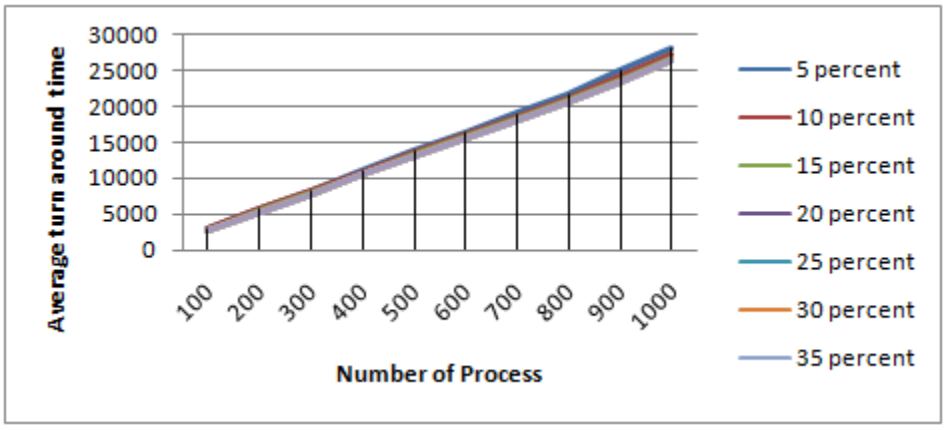

Figure 4.3: Percentage (%) comparison using Average turnaround time graph

The lines for 25, 30-50 (%) are not clearly shown because, they have the same value and their values are very close to that of 20% that is why they are hidden in the line for 20%. Any of these % above 20% makes the program to behave like shortest burst first scheduling. This is because, there will be no CS. Meaning, all the processes will run to completion in the first round and, this is not a good practice in RR scheduling. Any of these percentages below 20% increases the number of CS. Meaning, very few processes will benefit from the % and, the more the CS the worst the performance.

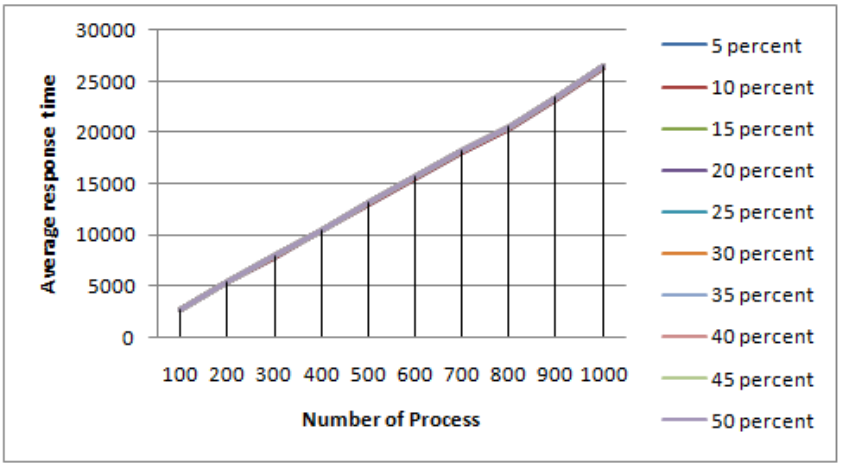

Figure 4.4: Percentage (%) comparison using Average response time graph

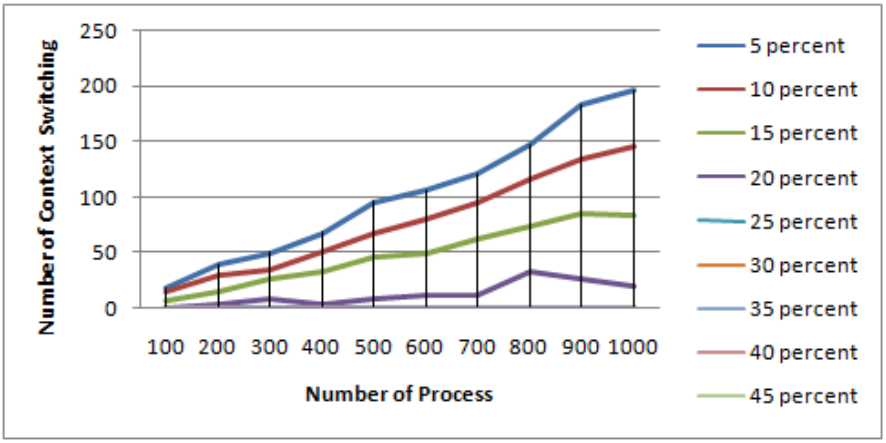

Figure 4.5: Percentage (%) comparison using Number of context switching graph

The graph of Figure 4.4 for ART shows that the response time for 5% is better. Meaning, the smaller the % the better the response time. But, the graph clearly shows that the difference between the percentages are very small that is why the lines condense in one place. Therefore, for the ART, any percentage you choose can serve since their difference is insignificant. There for, we can go for the 20%.

The graph of Figure 4.5 for NCS shows that 20% is the best. The lines for other percentages above it all lie on the X-axis because their CS is 0 throughout, which is also not a good practice in RR scheduling, at least, some processes must go for second round. The lines for other percentages below the 20% must have CS more than the 20% which will affect the performance.

4.9 PRESENTATION OF RESULTS.

The results obtained are presented in a tabula form. When the simulator is run for 10 times with number of processes starting from 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000,the simulator is able to find the average waiting time, average turnaround time, number of context switch and average response time. The average response time was only taken for the first round.

Table 4.1: Classical RR Simulation Result

CLASSICAL RR RESULT TABLE				
No of Process	***AWT(m/s)***	***ATAT(m/s)***	***ART(m/s)***	***NCS***
100	8219.88	8342.8	5326.75	48
200	17301.9	17428.74	11188.52	93
300	24634.49	24760.05	16493.91	132
400	33272.73	33397.72	22044.66	178
500	41061.43	41188.28	27594.68	220
600	50753.51	50880	33203.95	276
700	56408.49	56532.4	37949.94	303
800	65771.78	65896.15	43478.94	362
900	76383.82	76511.96	50249.62	413
1000	86032.86	86159.52	56042.78	470

Table 4.2: DRRCP simulation result

DRRCP RESULT TABLE				
No of Process	***AWT(m/s)***	***ATAT(m/s)***	***ART(m/s)***	***NCS***
100	8045.26	8168.18	5333.16	44
200	16298.1	16424.94	11228.78	77
300	23849.96	23975.52	16526.15	119
400	32179.78	32304.78	22079.82	159
500	36940.83	39767.68	27643.86	198
600	49360.24	49486.74	33255.03	252
700	54929.41	55053.32	37987.64	276
800	63483.17	63607.54	43555.74	322
900	73281.98	73410.11	50360.05	367
1000	81402.06	81528.72	56208.13	393

Notice the following points in Table 4.1 and 4.2.

If you look at the results obtained in these tables, they tempt to be a little bit closer to each other and still, each of the corresponding parameter values in the two tables when compared show that DRRCP is a direct improvement of the classical RR. Though its response time is not as good as that of the classical RR.

Table 4.3: HLVQTRR simulation result

HLVQTRR RESULT TABLE				
No of Process	***AWT(m/s)***	***ATAT(m/s)***	***ART(m/s)***	***NCS***
100	9135.16	9258.08	3055.88	100
200	19029.19	19156.03	6408.32	200
300	28231.03	28356.59	9458.54	300
400	37565.38	37690.38	12628.03	400
500	47364.52	47491.37	15721.42	500
600	57050.23	57176.72	19162.72	600
700	65084.95	65208.86	21773.25	700
800	74633.64	74758.02	24941.02	800
900	86355.07	86483.21	28759.61	900
1000	95149.46	95276.13	31877.16	1000

Notice in Table 4.3, the number of process corresponds to the number of context switching. This should logically affect the AWT and ATAT.

Table 4.4: IRR simulation result

IRR RESULT TABLE				
No of Process	***AWT (m/s)***	***ATAT (m/s)***	***ART (m/s)***	***NCS***
100	6080.76	6203.68	6080.76	0
200	12770.16	12897	12770.16	0
300	18843.6	18969.16	18843.6	0
400	25149.18	25274.17	25149.18	0
500	31310.62	31437.46	31310.62	0
600	38179.44	38305.94	38179.44	0
700	43368.42	43492.33	43368.42	0
800	49691.1	49815.47	49691.1	0
900	57292	57420.14	57292	0
1000	63508.2	63634.87	63508.2	0

Notice the following point in Table 4.4,

Context switching: this is zero for any number of runs. This means that none of the processes went beyond first round, in each run of the simulator (which is not a good practice in RR. Because, in any giving RR scheduling at least some processes are expected to be switched to second round).The reason behind this problem is due to the time quantum chosen, which is the average of the processes burst time and the condition impose on this algorithm. If a very small time quantum is used, say 10 or 20, some processes may go for second round if the burst time of these processes are very large.

Table 4.5: IMRRSJF simulation result

IMRRSJF RESULT TABLE				
No of Process	***AWT (m/s)***	***ATAT (m/s)***	***ART (m/s)***	***NCS***
100	5361.97	5484.89	5075.02	19
200	11331.18	11458.02	10519.31	44
300	16763.83	16889.39	15526.15	66
400	22308.95	22433.94	20751.27	86
500	28486.22	28613.06	26306.66	113
600	34349.87	34476.37	31340.91	146
700	38891.96	39015.87	35921.55	157
800	44829.86	44954.24	41226.09	185
900	52123.45	52251.58	47869.41	211
1000	56694.77	56821.43	52996.29	208

Table 4.6: MMRRA simulation result

MMRRA RESULT TABLE				
No of Process	*AWT (m/s)*	*ATAT (m/s)*	*ART (m/s)*	*NCS*
100	5096.76	5219.68	5094.80	2
200	10600.82	10727.66	10573.39	8
300	15644.98	15770.54	15631.32	7
400	20938.85	21063.84	20845.94	20
500	26516.07	26642.92	26462.93	17
600	31611.57	31738.07	31549.65	20
700	36240.44	36364.35	36119.28	30
800	41556.82	41681.19	41471.55	27
900	48205.58	48333.71	48179.30	16
1000	53280.45	53407.12	53269.60	11

Notice the following points in Table 4.5 and 4.6.

If you look at the results obtained in these two tables, they tempt to be a little bit closer to each other and they have a better AWT, ATAT and CS compared to other algorithms. This is due to the fact that processes were sorted on Shortest Job First and the scheduling is done.

Finally, if you look at these two tables ones again, you will find out that MMRRA use to be the best in most of the results, this is due to the method used.

4.10 DISCUSSION OF RESULTS.

The graphs of Figure 4.6 - 4.9 were gotten from the above corresponding results Tables. Each graph compared the Six algorithms on a single criterion fetched from each of the four tables.

Figure 4.6 compared only the average waiting time for the six algorithms. From the graph, it is clear that the result for MMRRA is the best followed by IMRRSJF. IRR is the next followed by DRRCP and CLASSICAL whose result are very close though DRRCP shows better result than the CLASSICAL. HLVQTRR is to be considered having the worst result.

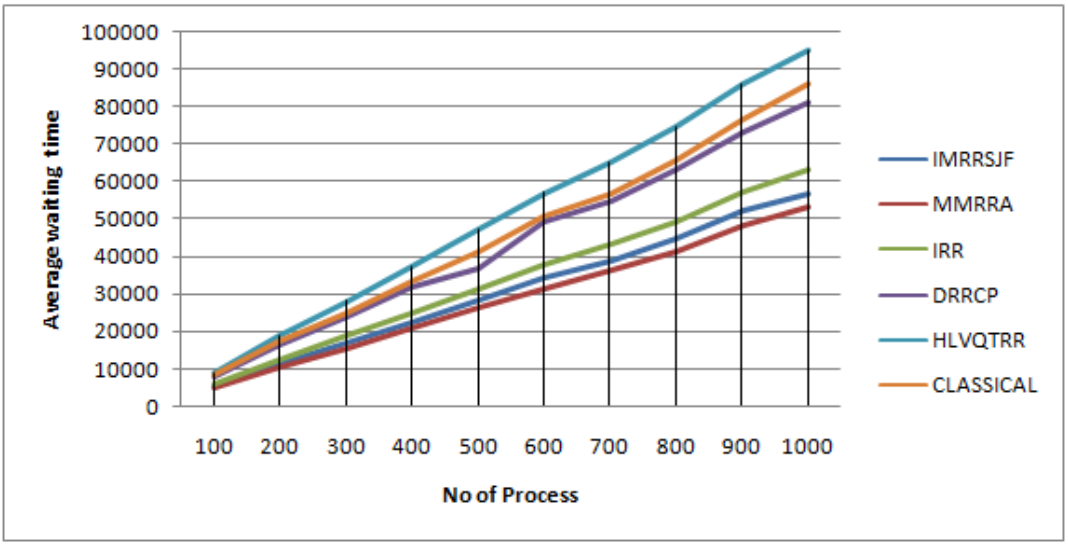

Figure 4.6: Average waiting time graph

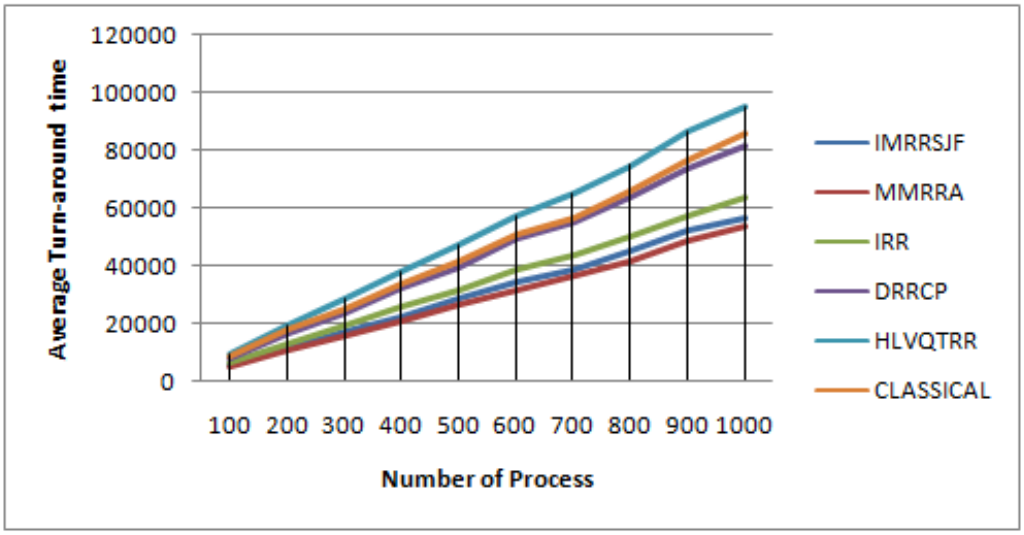

Figure 4.7: Average turnaround time graph

Figure 4.7 compared only the average turnaround time for the six algorithms. From the graph, it is clear that the result is similar to that of average waiting time. That is, MMRRA is the best followed by others as in the case of average waiting time..

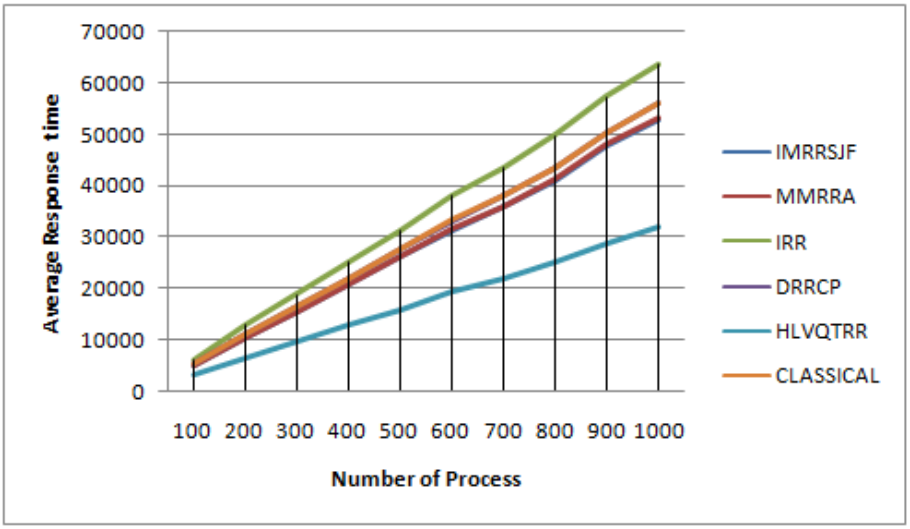

Figure 4.8: Average response time graph

Figure 4.8 compared only the average response time for the six algorithms in the first round only. From the graph, it is very clear that HLVQTRR has the best result. If you look at MMRRA and IMRRSJF they have an average result and their results are too close that is why the line for IMRRSJF is not shown clearly it is hidden in MMRRA next is the CLASSICAL and DRRCP that show similar situation to that of MMRRA and IMRRSJF. Meaning, the line for DRRCP is also hidden in the CLASSICAL. IRR has the worst result.

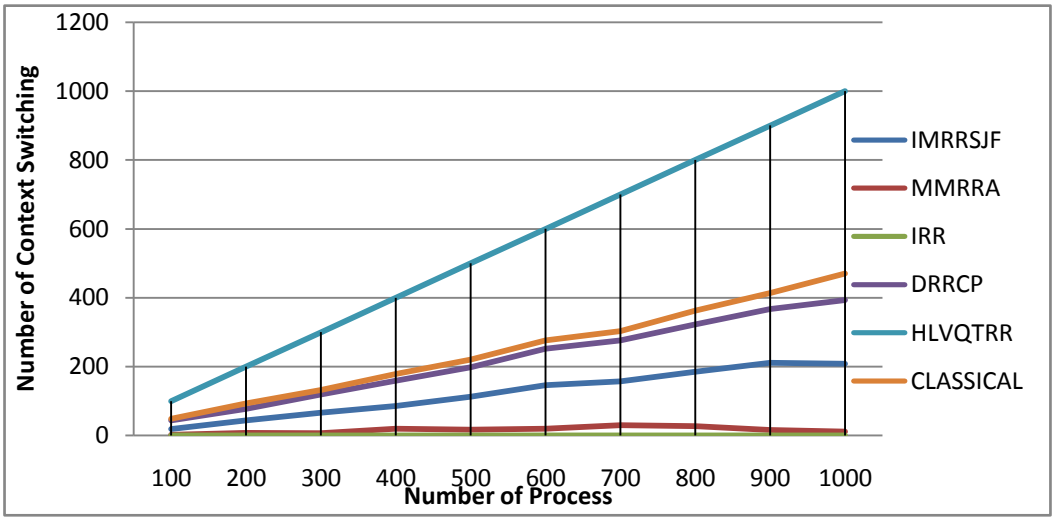

Figure 4.9: Number of context switching graph

Figure 4.9 compared only the number of context switching for the six algorithms. From the graph, it shows that an extreme algorithm is the HLVQTRR which has much context switching. This is because, in every running of the program, its number of context switching is equal to the number of processes, this is due to the fact that each process must undergo one context switching.

On the other hand, IRR has the lowest and at the same time the worst context switching. It is the worst because in any giving RR scheduling at least some processes are expected to be switched to second round. In this case, none was switched to the next round. The context switching of IRR is set to be 0, in all running of the program. This is the reason why the line graph of the IRR is not shown clearly in the graph because it lies directly on the x-axis of the graph. (The reason behind this is the time quantum that we choose to be the average of the processes burst time. If we choose time quantum to be very small say 10 or 20, there may be possibility of having some processes going for second round).

The best algorithm in terms of context switching is the MMRRA followed by IMRRSJF. DRRCP and CLASSICAL have an average result.

CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 SUMMARY

Round Robin (RR) CPU scheduling algorithm has found its chief use in time sharing system. The greatest challenge in RR is Quantum time. That is, what should be the Quantum time? 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 reduce the number of context switching, average waiting time and average turnaround time, we must find a dynamic Quantum time. This will provide a Quantum time that change automatically according to a circumstance so as to improve the general performance of the system. In this regard, an enhanced versions of RR CPU scheduling algorithms which is dynamic in nature was proposed. That is, Modified Median Round Robin Algorithm(MMRRA). MMRRA was designed to provide a mechanism to prevent the system from performing unnecessary context switching. In an attempt to solve unnecessary context switching, the average waiting time, average turnaround time and number of context switching were greatly improved as well. Using exponential distribution function, random variables were generated to evaluate MMRRA against the DRRCP, HLVQTRR, Classical RR, IMRRSJF and IRR. These six algorithms were compared and evaluated using both the analytic and simulation techniques. MMRRA was designed to achieve a better average waiting time, a better average turnaround time and a better context switching than any of the above algorithms, and it does. In terms of average response time, it has an average performance.

One thing that must be put into consideration is that, as stated in chapter three by Flynn and McHoes (2001) “if the system favors one type of user then it hurts another or doesn’t efficiently use its resources”. What this statement means is that: in trying to make a system do what you want, you can decide to sort which will incur sorting cost, decide to use priority on shorter jobs, decide to jerk up quantum time above average or even decide to impose some calculations that will attract cost expectation. All these can be applied but bearing in mind that the benefit gained is more than the drawback and the final decision rests with the system designer, who must determine which criteria are most important for that specific system. So you would select the scheduling policy that most closely satisfied your criteria.

5.1.1 Advantage of MMRRA over IMRRSJF

For IMRRSJF, there is a contradiction in their QT usage. Their Flow Chart and Algorithm show that QT is recalculated in each round of the scheduling. If what they said is true, that means the recalculation of the QT will incurs cost of computation which should be avoided. On the other hand, in their illustration, they stated that QT is calculated only once.

This contradiction makes the researcher to critically analyse this paper by solving their problems using these two methods. I came up with a conclusion that they used only one QT for each round. This shows that their method is not really a dynamic RR as they said but a Classical RR using shortest burst first approach.

For MMRRA, the calculation of quantum time is once. For the second round of this algorithm, the processes must run to completion because the QT for this round is their remaining burst time.

Which is an advantage. Also, MMRRA is better than IMRRSJF in: AWT, ATAT & NCS. Apart from being better, the MMRRA is a dynamic RR while IMRRSJF is a static one. Which is another advantage.

5.1.2 Advantage of MMRRA over HLVQTRR

HLVQTRR considers only degree of multiprogramming and response time. It ignores the other criteria. The writer in his work stated that:

"Multiprogramming tracks those processes that went beyond the first round" Looking at this statement, it shows that multiprogramming is nothing but number of context switching. Going to his statement, it means, the better the multiprogramming the higher the context switching.

MMRRA clarifies "degree of multiprogramming" and how to access number of context switching.

A research made on degree of multiprogramming described the concept as:

I. The maximum number of processes that a single-processor system can accommodate efficiently (gmu; 1997).

II. Medium term scheduling is part of the swapping. It removes the processes from the memory. It reduces the degree of multiprogramming (tutorialspoint; 2016).

III. The concept of multiprogramming is used for improving the CPU utilization. When there are multiple processes running then CPU utilization can be calculated by the formula: CPU utilization = 1 – p^n where, p is the time spend by a process and n is the number of processes simultaneously in memory. Here, n is also known as degree of

multiprogramming (moreprocess; 2013)

This shows that Simon's definition on multiprogramming is nothing but the **number of Context switching**, which should be minimised.

Simon, further stated that: "The major emphasis of his scheduling is: multiprogramming, response time, simplicity, elimination of unnecessary context switching". But, if you look at this statement, in a situation where you have process with a very small burst time, applying HLVQTRR will result to unnecessary CS on the process. Meaning, it's better to allow the process run to completion than allowing it to go for the second round.

Nevertheless, MMRRA is better than HLVQTRR on three criteria (AWT, ATAT and NCS). HLVQTRR is better than MMRRA in only one criterion which is ART. But still the result for the MMRRA on the response time is average which is not bad.

5.1.3 Advantage of MMRRA over DRRCP

The concept of DRRCP is not suitable to be a RR scheduling algorithm on its own because it has some lapses with respect to the four criteria. It is only suitable to incorporate it in another algorithm just as the writer stated:

"Performance of DRRCP >= Performance of classical RR". He further stated that:

"The justification for this representation is that depending on the dataset, DRRCP may work purely as classical RR. On a final note, the concept of DRRCP algorithm can fit into any of the proposed dynamic RR CPU scheduling algorithms".

MMRRA is far better than DRRCP because, MMRRA is not a slight improvement of classical RR. It is an algorithm standing on its own which performs better than
both the DRRCP & Classical RR in both the four criteria.

5.1.4 Advantage of MMRRA over IRR

For IRR it does not have any specific QT. Any value can be taking as QT but bearing in mind that if the remaining burst of any process <the quantum time chosen you should allow that process to run to completion.

In reality, if we take a small value as our QT say 20ms, it is more likely to have more CS if the processes burst time are very high. Which will result to bad performance.

On the other hand, if we choose QT to be larger say the average of the processes burst time, most of the processes will run to completion due to the set condition. This will make the algorithm behave like first come first serve.

MMRRA controls all these problems by taking a reasonable QT with a reasonable condition. It's also better than the IRR in all the criteria only in NCS that IRR use to be 0 always because QT is taken to be the average of the processes burst time. This is not a good practice in RR algorithm.

5.1.5 Some Major Challenges of DYNAMIC RR Algorithms.

The first issue is the jerking up of quantum time above average. Jerking of quantum time above average 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. Continue 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. If we look at MMRRA and IMRRSJF, their Time Quantum depend on the processes being generated. Sometimes it can be more than or equals to the average of the processes burst time. For our simulation, If we take for example Mean=200 and Deviation=5, the processes burst time that will be generated will be close to each other which will almost gives us the same TQ with some of the algorithms as the average of the processes burst time.

Apart from jerking up the quantum time, some incurred the cost of sorting. Sorting is an overhead and as much as possible it should be avoided. And it is very clear that the sorting also brings about priority to shorter jobs. Which it said to be a disadvantage.

MMRRA is one of the victims here. But, if we look at the situation, whatever method we choose it has its own advantages and disadvantages. But, what we are trying to impose is to make the advantages be higher than the disadvantages. Now, if we look at the Average Turnaround Time and the Average Waiting Time graph, it is very clear that the MMRRA and IMRRSJF(the sorted algorithms) perform far much better than the unsorted HLVQTRR, DRRCP etc. At the same time if we look at the graph of Average Response Time, though it is clear that HLVQTRR(an unsorted algorithm) has the best result. But still, MMRRA and IMRRSJF(the sorted algorithms) have an average result that is better than DRRCP, CLASSICAL and IRR(unsorted algorithms). Therefore, the advantages of the sorted algorithms here are more than the disadvantages.

5.1.6 Critisism With Respect To Context Switching On Most Of the proposed Round Robin Algorithms reviewed

Interrupt cause the operating system to change a CPU from its current task and to run a kernel routine. Such 0perations happen frequently on general-purpose systems. When an interrupt occurs, the system needs to save the current context of the process currently running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it. The context is represented in the PCB of the process; it includes the value of the CPU registers, the process state and memory-management information. Generically, we perform a state save of the current state of the CPU, be it in kernel or user mode, and then a state restore to resume operations.

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch. Context-switch time is pure over head, because the system does no useful work while switching
(Silberschatz *et al*., 2005).

Let us suppose that a process has been taken up for execution now. The execution cannot go beyond that time slice. This process may either finish up its execution before the time goes off or CPU will be preempted from it after the timer goes off and this cause an interrupt to the operating system. At this time, context switching will take place. The next process will be taken up from the ready queue. The process, which is left by the CPU, will be added to the tail of the ready queue.

Therefore, from these points discussed above, it is very clear that context switching occur only if process is preempted from running. But, if a process runs to completion and leave the queue, we do not call this a context switching. Contrary to these statements, many proposed work has been reviewed, but the way they use to view context switching is switching from one process to the other whether the process run to completion or not. The following are among the research works reviewed on this issue. Ranging from 2009-2014. Matarneh (2009), Behera *et'al* (2010), Hiranwal and Roy(2011), Saxena and Agarwal (2012), Negi (2013), Agha and Jassbi (2013), Raman and Mittal (2014) and Simon *et'al* (2014).

If you access the Gantt charts or time line of these research works, you will realize that their number of context switching is nothing but the number of switching on the time line whether a process run to completion or not. Whereas, Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.

5.2 CONCLUSION

Conclusively, this research work is set to provide a better solution to the classical RR, IRR, IMRRSJF, HLVQTRR and DRRCP CPU scheduling algorithms. In this regard, The Modified Median Round Robin Algorithm (MMRRA) is designed. From the analysis, it has become clear that MMRRA has provided some significant solutions to the shortcomings of classical RR and most of the proposed RR algorithms that were reviewed. MMRRA is set to minimize no of context switching (CS) such that it does not suffer much from average waiting time and average turnaround time . This has been achieved. It is also set to provide control to unnecessary context switching. This in turn increases the performance of the system by also minimizing average waiting time, average turnaround time and number of context switching. Generally in Round

Robin algorithm the performance depends upon the size of fixed or static Time Quantum (TQ). If TQ is too large then Round Robin algorithm approximate to First Come First Served (FCFS). If the Time Quantum is too small then there will be many context switching between the processes. So, our approach solved these problems by taking a dynamic TQ that will neither be First come First Serve nor, increase the number of context switching.

5.3 RECOMMENDATION

Since MMRRA has proven to be the better algorithm, it will be very interesting to evaluate these algorithms using implementation technique. It may cost much, but it worth it.In addition, what happened if this algorithm is tried on high performance computers in which multiple processors are used instead of a single processor? Also, what happened when processes' arrival time is considered? These are good areas to be researched upon.

REFERENCES

Abur, M. Muhammed, A. Danjuma, S. and Abdullahi, S.(2011). A Critical Simulation of CPU Scheduling Algorithm Using Exponential Distribution. (IJCSI) International Journal of Computer Science Issues, Vol. 8, Issue 6, No 2, ISSN (Online): 1694-0814.pp 201-206

Agha A.E.A and Jassbi S.J (2013).*A New Method to Improve Round Robin Scheduling Algorithm with Quantum Time Based on Harmonic-Arithmetic Mean*.modern education and computer science PRESS DOI:10,5815/ijitcs. pp. 56-62

Banerjee, P., Banerjee, P. and DhalS.S. (2012a) *Comparative Performance Analysis of Average Max Round Robin Scheduling Algorithm 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.

Banerjee, P., Banerjee, P. and DhalS.S. (2012b). *Performance Evaluation of a New Proposed Average Mid Max Round Robin Scheduling Algorithm with Round Robin Scheduling Algorithm*, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, ISSN:2277 128X. pp.143-151.

Banks J. S., Carson J.S., Nelson B. L., Nicol D.M., (2005).Discrete-Event System Simulation Fourth Edition.

Behera H.S., Mohanty R. and Nayak D. (2010).*A New Proposed Dynamic Quantum With Re-Adjusted Round Robin Scheduling Algorithm and its Performance Analysis*.

International journal of computer applications (0975-8887) volume 5. pp. 10-15

Behera, H.S., Swain B.K, Parida A.K and Sahu G.(2012). *A New Proposed Round Robin with Highest Response Ratio Next Scheduling Algorithm for Soft Real Time System*. (IJEAT) International journal of engineering and advanced technology. ISSN: 2249-8958, Volume 1, pp.200-206.

Brownen (2003) Simple Queuing Models (https://www.cs.utexas.edu/users/browne/cs380ns2003/Papers/SimpleQueuingModelspdf.pdf) accessed on march 4th 2016.

Finkel, R. (1986). *An operating systems vade mecum*. Englewood Cliffs, NJ: Prentice Hall.

Flynn, I. M. and McHoes A. M. (2001).*Understanding Operating Systems (3rd ed.).* pp. 462

Hiranwal S. and Roy K.C. (2011).Adaptive Round Robin Scheduling using Shortest Burst Approach Based on Smart Time Slice. International Journal of Data Engineering (IJDE), Volume 2, pp. 237-243.

Ibrahim, A. (2012) Process Scheduling In Longest Job First (LJF) Algorithm. A proposed framework for Starvation Problem. A thesis from Department of Mathematics, Ahmadu Bello University, Zaria. Nigeria

Linfo (2006) *Context Switch Definition* (http://www.linfo.org/context_switch.html) accessed on June 17, 2015.

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 Science 6(10): pp1831-1837 ISSN 1546-9239.pp. 1831-1837

Mishra, M.K. and Khan A.K.(2012). *An Improved Round Robin CPU Scheduling Algorithm*. (JGRCS) Journal of global research in computer science. Volume 3, ISSN: 2229-371X. pp.64-69.

Mohanty R., Das M., Prasanna M.L. and Sudhashree (2011).*Design and Perormance Evaluation Of a New Proposed Fittest Job First Dynamic Round Robin Scheduling Algorithm*. International journal of computer information systems.volume 2, ISSN:2229-52 pp. 23-27

Negi S. (2013). *An Improve Round Robin Approach using Dynamic Time Quantum for Improving Average Waiting Time*. International journal of computer application (0975-8887) volume 69. pp. 12-16

Noon, A., Kalakech, A. and Kadry, S.(2011). *A New Round Robin Based Scheduling Algorithm for operating systems: Dynamic Quantum using the mean average*. IJCSI *International Journal of Computer Science Issues*, Vol. 8, Issue 3, pp. 224-229.

OPRE (6301) Importance Probability distributionshttps://www.utdallas.edu/~scniu/OPRE-6301/documents/Important_Probability_Distributions.pdfaccessed on March 5th 2016

RakeshMohanty, Manas Das, M. Lakshmi Prasanna and Sudhashree (2011). Design and performance evaluation of a New Proposed Fittest Job First Dynamic Round Robin Scheduling Algorithm.*Iternational Journal of Computer Information Systems* Vol2, No.2 pp 23-93.

Raman and Mittal, P. K. (2014). An Efficient Dynamic Round Robin CPU Scheduling Algorithm.International journal of advanced research in computer science and software Engineering. Volume 4,ISSN: 2277 128X. pp. 906-910

Saxena F.A.H and Agarwal S.B.P (2012).*Design and Performance Evaluation of Optimum Service Time Concept For Round Robin Algorithm*. International journal of machine learning and computing, vol 2, pp. 113-117.

Shyam, R. and Nandal, S. K. (2014).*Improved Mean Round Robin with Shortest Job First Scheduling.*international journal of advanced research in computer science and Software engineering. Volume 4,ISSN:2277-128X. pp. 170-179.

Silberschatz A, Galvin P.B. and Gagne .G, (2005), *Operating Systems Concepts*, (7thed), John Wiley and Sons, USA. pp. 1-886.

Simon A. Abdullahi S. and Junaidu S.(2014a). *Dynamic Round Robin with Controlled Preemption*. (IJCSI) international journal of computer science Issues, vol 11, Issue 3, ISSN 1694-0784. pp. 109-117

Simon A. Abdullahi S. and Junaidu S.(2014b). *Half Life Variable Quantum Time Round Robin.* (IJCSIT) international journal of computer science and information technologies, vol.5 (6), ISSN: 0975-9646 pp. 7210-7217.

Tanenbaum, A. S. (2001). Modern Operating Systems (2nd ed.). Prentice - Hall, Inc. pp. 951

Appendix : Source code

```
Public Function CheckExist()
'On Error Resume Next
Dim confirm As String
Dim rs As New ADODB.Recordset
Dim con As New ADODB.Connection
Dim sCmd As String
'create connection to database
con.CursorLocation = adUseClient
sCmd = "Provider = Microsoft.Jet.OLEDB.4.0;; persist security info = false;"
sCmd = sCmd & " data source = " & App.Path & "\" & "test.mdb"
Set con = New ADODB.Connection
   With con
      .ConnectionString = sCmd
      .Open
   End With
   'insert into database
rs.Open "INSERT INTO tblTest(Digit,pte) values('" & Text1.Text & "','" & Text13.Text & "')",
con, 2, 3
Adodc1.Refresh
End Function
'FIND THE AVERAGE OF GENERATED NUMBER
```

```
Public Function findsum()
'On Error Resume Next
'create connection to database
Dim rs As New ADODB.Recordset
Dim con As New ADODB.Connection
Dim sCmd As String
Dim total As Integer
con.CursorLocation = adUseClient
sCmd = "Provider = Microsoft.Jet.OLEDB.4.0;; persist security info = false;"
sCmd = sCmd & " data source = " & App.Path & "\" & "test.mdb"
Set con = New ADODB.Connection
   With con
      .ConnectionString = sCmd
      .Open
   End With
   'find average
 rs.Open "Select sum(Digit)as total from tblTest", con, 2, 3
       Text2.Text = Format(rs!total / Val(txtp.Text), 0)
End Function
'DELETE TEMPORARY DATASET
Public Function delExist()
'On Error Resume Next
Dim confirm As String
```

```
'create connection to database
Dim rs As New ADODB.Recordset
Dim con As New ADODB.Connection
Dim sCmd As String
con.CursorLocation = adUseClient
sCmd = "Provider = Microsoft.Jet.OLEDB.4.0;; persist security info = false;"
sCmd = sCmd & " data source = " & App.Path & "\" & "test.mdb"
Set con = New ADODB.Connection
   With con
      .ConnectionString = sCmd
      .Open
   End With
'delete from database
rs.Open "Delete * from tblTest", con, 2, 3
Adodc1.Refresh
End Function
'LOAD DATA FROM DATABASE ALREADY SORTED
Public Function loadlist()
'create connection to database
Dim rs As New ADODB.Recordset
Dim con As New ADODB.Connection
Dim sCmd As String
Dim fname, SQL As String
```

```
Dim statement As String
con.CursorLocation = adUseClient
sCmd = "Provider = Microsoft.Jet.OLEDB.4.0;; persist security info = false;"
sCmd = sCmd & " data source = " & App.Path & "\" & "test.mdb"
Set con = New ADODB.Connection
With con
.ConnectionString = sCmd
.Open
End With
'sort and retrieve from database
   statement = "SELECT * FROM tblTest ORDER BY Digit*1 ASC "
   Set rs = con.Execute(statement, , adCmdText)
   Do While Not rs.EOF
      List3.AddItem rs!Digit
      List14.AddItem rs!pte
      List15.AddItem rs!Digit
      rs.MoveNext
   Loop
   rs.Close
   con.Close
End Function
'FIND TIME QUANTUM
Public Function findtq()
```

```
Dim highest, q, r, a, b, c As Currency
Dim d, e, z As Integer
q = Val(txtp.Text)
r = List3.ListCount
z = (r / 2)
'use median
'highest = Format(Sqr(Val(Text2.Text) * List3.List(q - 1)), 0)
If r Mod 2 = 0 Then
Text10.Text = List3.List(z)
Text11.Text = List3.List(z - 1)
Text9.Text = Round(((Val(Text10.Text) + Val(Text11.Text)) / 2), 2)
   Else
Text9.Text = (List3.List(z))
End If
highest = Format(Sqr((Val(Text9.Text)) * List3.List(q - 1)), 0)
Text3.Text = highest
Label6.Caption = highest
Dim p, k As Integer
k = 0
For p = 0 To List3.ListCount - 1
    'calculate the quantum remember to increase with 10 percent
      If highest > List3.List(p) Then
       'If highest - List3.List(p) > (List3.List(p) / 10) Then
```

```
        timeq.AddItem List3.List(p)
    ElseIf List3.List(p) - highest < (List3.List(p) / 10) Then
        timeq.AddItem List3.List(p)
    Else
        timeq.AddItem highest
    End If
     List23.AddItem highest
Next
End Function
Public Function firstround()
Dim p, k, cs, multi As Currency
cs = 0
multi = 0
For p = 0 To timeq.ListCount - 1
k = List3.List(p) - timeq.List(p)
    If k > (List3.List(p) / 10) Then
        List4.AddItem List3.List(p) - k
        List5.AddItem k
    Else
        List4.AddItem List3.List(p)
        List5.AddItem "0"
    End If
    'compute context switching
```

```
  If List5.List(p) <> 0 Then
        cs = cs + 1
    End If
  'compute degree of multi programming
    If List4.List(p) <> 0 Then
        multi = multi + 1
    End If
Next
txtmulti.Caption = cs
txtcs.Caption = (multi + cs) - 1
End Function
Public Function starts()
Dim p, k, art As Currency
k = 0

For p = 0 To (List3.ListCount - 1)
      List6.AddItem k
      k = k + timeq.List(p)
  Next
List6.AddItem k, Val(txtp.Text)
End Function
'AVERAGE RESPONSE TIME
Public Function aresponsetime()
```

```
Dim p, k, art As Currency
k = 0
 art = 0
For p = 0 To ((List6.ListCount - 1) - 1)
        art = art + List6.List(p)
  Next
txtart.Caption = Round((art / (Val(txtp.Text))), 2)
End Function
'AVERAGE TURNAROUND TIME
Public Function tatime()
Dim p, k, atat, tats, q As Currency
k = 0
tats = 0
For p = 0 To timeq.ListCount - 1
k = k + Val(List4.List(p))    '+ Val(List5.List(p))
   List8.AddItem k
 'tats = tats + ListTat.List(p)
Next

q = List8.List(List8.ListCount - 1)
Text6.Text = q

For p = 0 To List8.ListCount - 1
```

```
 If List5.List(p) <> 0 Then


 k = q + Val(List5.List(p))
' k = k + Val(ListTat.List(p))
   ListTat.AddItem k
 'tats = tats + ListTat.List(p)
 q = k
 Else
 ListTat.AddItem List8.List(p)
End If
tats = tats + ListTat.List(p)
Next
If ListTat.ListCount - 1 = 0 Then
txtatat.Caption = tats
Else
atat = tats / ((ListTat.ListCount - 1) + 1)
txtatat.Caption = Round(atat, 2)
End If
End Function


'AVERAGE WAITING TIME
Public Function waiting()
On Error Resume Next
```

```
Dim p, k, q, wts, awt As Currency
For p = 0 To ((List6.ListCount - 1) - 1)
   k = Val(List6.List(p)) ' + Val(List5.List(p))
       List11.AddItem k
 'wts = wts + Listwt.List(p)
Next
'q = List12.List(List6IMRR.ListCount - 1)
  q = List6.List(List6.ListCount - 1)
For p = 0 To List6.ListCount - 1
If List5.List(p) <> 0 Then
 'note
  k = q + Val(List5.List(p))
' k = k + Val(ListTat.List(p))
  ' ListwtIMRR.AddItem k
   List19.AddItem k
   List20.AddItem k
 'tats = tats + ListTat.List(p)
 q = k
 Else
' ListwtIMRR.AddItem List12.List(p)
   List19.AddItem List11.List(p)
   List20.AddItem 0
End If
```

```
wts = wts + List19.List(p)
'wts = wts + ListwtIMRR.List(p)
Next
List20.AddItem List6.List(List6.ListCount - 1)
For p = 0 To List6.ListCount - 1
   aa = Val(List6.List(p))  ' + Val(List5.List(p))
   bb = Val(List20.List(p))
   cc = Val(List6.List(p + 1))
If List5.List(p) <> 0 Then
      dd = aa + (bb - cc)
        Listwt.AddItem dd
   '     ff = Val(txtp.Text) - 1
'ee = Val(List6IMRR.List(ff)) + (List16.List(ff) - List17.List(0))
Else
' ListwtIMRR.AddItem List12.List(p)
   Listwt.AddItem aa
End If
'wts = wts + List16.List(p)
'wts = wts + ListwtIMRR.List(p)
Next
'ee = List17.List(0) + (bb - cc)
'ListwtIMRR.AddItem ee, ff
```

```
End Function
'FILL RESULT TO LISTBOX
Public Function fill()
Dim p, k, wts, awt As Currency
wts = 0
For p = 0 To List3.ListCount - 1
  ListSimrar.AddItem ("Process  " & List14.List(p) & vbTab & List3.List(p) & vbTab & List23.List(p) & vbTab & List4.List(p) & vbTab & List5.List(p) & vbTab & List6.List(p) & vbTab & Listwt.List(p) & vbTab & ListTat.List(p))
   wts = wts + Listwt.List(p)
Next
awt = wts / ((Listwt.ListCount - 1))
txtawt.Caption = Round(awt, 2)
End Function
Public Function sorter()
Dim i As Currency
List3.Clear
      For i = List2.ListCount - 1 To 0 Step -1
         List3.AddItem (List2.List(i))
      Next
End Function
'CLEAR LISTBOXES
```

```
Sub clearsim()
List1.Clear
List2.Clear
List3.Clear
List4.Clear
List5.Clear
List6.Clear
ListTat.Clear
Listwt.Clear
ListSimrar.Clear
End Sub
```