A SCHEDULING ALGORITHM FOR MINIMIZING THE PENALTY RATIO VARIANCE

BY

OKPOKO, Onyinye Vivian

DEPARTMENT OF MATHEMATICS,

FACULTY OF SCIENCE,

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

MAY, 2015

A SCHEDULING ALGORITHM FOR MINIMIZING THE PENALTY RATIO VARIANCE

BY

OKPOKO, Onyinye Vivian

MSc/SCIE/10581/2010-2011

A Thesis submitted to the School of Postgraduate Studies, Ahmadu Bello University, Zaria

In partial fulfilment of the requirements for the Award of

Masters Degree in Computer Science

Department of Mathematics, Faculty of Science

Ahmadu Bello University, Zaria, Nigeria

May, 2015

Declaration

I hereby declare that the work in this thesis titled “A Scheduling Algorithm for Minimizing the Penalty Ratio Variance” was performed by me in the Department of Mathematics, under the supervision of Prof. S. B Junaidu and Dr. A.A Obiniyi.

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.

OKPOKO, Onyinye Vivian _____ _____

Name of Student Signature Date

Certification

This thesis titled “A Scheduling Algorithm for Minimizing the Penalty Ratio Variance” 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.

Prof. S. B Junaidu	_____	_____
Major Supervisor	(Signature)	(Date)
Dr. A.A. Obiniyi	_____	_____
Minor Supervisor	(Signature)	(Date)
_____	_____	_____
External Examiner	(Signature)	(Date)
Dr. B. Sani	_____	_____
Head of Department	(Signature)	(Date)
Prof A. Z. Hassan	_____	_____
Dean, School of Postgraduate	(Signature)	(Date)

Dedication

This research is dedicated to my wonderful family and to God, The Almighty.

Acknowledgement

All glory belongs God, The Almighty and Lord of everything. I cannot thank you enough for giving me the gift of life and health to make this work a success. This work is all yours Lord.

My wholehearted appreciation and immense gratitude goes to my Major Supervisor, Professor S.B Junaidu for diligently going through this work, despite his very busy schedule, to make it a success.

I also give my unreserved gratitude to my lecturer and able Minor Supervisor, Dr. A.A Obiniyi for his indispensable input and for taking his time to give me all the guidance I needed.

I cannot fail to acknowledge the scholarly preparations given to me by my other Masters Degree instructors: Prof. O.S Adewale, Dr. M.B. Hammawa and Dr. S.E Abdullahi. In the same vein, I thank the Head of Department, Dr. B. Sani, the former Postgraduate Coordinator, Dr. A.M. Ibrahim and the current one, Dr. A. Yahaya: may God reward all of your efforts.

I also cannot fail to mention my wonderful family. My wholehearted appreciation and gratitude go to my parents, Engr. and Mrs. E.U. Okpoko for their endless support and words of encouragement throughout the period of my study here in ABU: God bless you abundantly. My love and gratitude also go to my siblings who have been so supportive of me, praying for me and encouraging me: you are the very best. I also acknowledge my uncle, Dr. J.I. Okpoko, for all his advice and direction.

I must say a big thank you to Chisom and Joy Azubuike- my sisters from another mother- for accommodating and putting up with me every time I was in Zaria: may God bless you. I cannot afford not to thank Mal. Alhassan Abdulkadir for his technical assistance. My acknowledgement and thanks also go to my colleagues and friends: Mrs. Gloria Shidali, Mrs. Toyin PeteRock Sadiq, Mrs. Esther Oludare, Mrs. Odunayo Ife-Aina and Glory Ononokpono: thanks for your encouragement and friendship.

ABSTRACT

Scheduling is the technique of deciding which process is given control of a computer resource at a particular time. Processes are numerous while resources- such as the central processing unit (CPU), bandwidth and memory among others - are scarce. It therefore becomes necessary for scheduling to be done. Since there is no optimal scheduling algorithm, an algorithm becomes suitable based on the scheduling criteria that the operating system is designed to uphold. This thesis focuses on the development and evaluation of the performance of a CPU scheduling algorithm that achieves service timeliness by minimizing the penalty ratio variance. The algorithm inherits properties from the round robin scheduling algorithm. The model was tested and evaluated for performance alongside the Round Robin algorithm and some modifications of it, and was found to perform better than these other algorithms on penalty ratio variance minimization. This therefore implies that the turnaround time of a process can be determined before time; and can be evaluated to be a multiple of its penalty ratio.

Table of Contents

Declaration ..... **i**
Certification ..... **ii**
Dedication ..... **iii**
Acknowledgement ..... **iv**
Abstract ..... **v**
Table of Contents ..... **vi**
List of Tables ..... **viii**
List of Figures ..... **ix**
List of Abbreviations ..... **x**
1.0 GENERAL INTRODUCTION **1**
1.1 Background **1**
1.2 Research Questions **3**
1.3 Motivation of the Study **3**
1.4 Aim of the Study **3**
1.5 Objectives of the Study **3**
1.6 Methodology **4**
1.7 Scope **4**
1.8 Significant Contributions **4**
1.9 Organization of the Thesis **4**
2.0 LITERATURE REVIEW **6**
2.1 Introduction to Operating Systems **6**
2.2 Processes **7**
2.3 Scheduling **10**
2.3.1 Long-term or admission scheduler 12
2.3.2 Mid-term or memory scheduler 12
2.3.3 Short-term or CPU scheduler 14
2.3.4 Performance criteria for scheduling algorithms (Scheduling criteria) 14
2.4 Scheduling Algorithms in Operating Systems **16**
2.4.1 First come first served (FCFS) 16
2.4.2 Shortest job first (SJF) 17

2.4.3 Round robin (RR) 19
2.4.4 Priority scheduling 20
2.4.5 Highest response ratio next (HRRN) 21
2.5 Related Work 22
3.0 MINIMIZED PENALTY RATIO VARIANCE FRAMEWORK 26
3.1 Introduction 26
3.2 Approach of the Model 27
3.3 The Minimized Penalty Ratio Variance Algorithm (MPRV) 28
3.3.1 The proposed model 29
3.3.2 MPRV algorithm 30
2.3.3 Flowchart of the proposed model 32
4.0 IMPLEMENTATION AND TEST OF THE MINIMIZED PENALTY RATIO VARIANCE ALGORITHM 33
4.1 Introduction 33
4.2 Case Assumption 33
4.3 System Requirements 33
4.3.1 Notepad++ 33
4.3.2 PHP Language 34
4.3.3 Simulation system specifications 34
4.4 Simulation Framework 34
4.5 Experiment Case Alongside Round Robin Algorithms 36
4.5.1 Classical round robin 37
4.5.2 Dynamic quantum re-adjusted round robin (DQRRR) 39
4.5.3 Improved round robin (IRR) 41
4.5.4 New improved round robin (NIRR) 43
4.5.5 Minimized penalty ratio variance algorithm 45
4.6 Results, Discussion And Conclusion 48
5.0 SUMMARY, CONCLUSION AND FUTURE WORK 54
5.1 Introduction 54
5.2 Summary 54
5.3 Conclusion 55

5.4 Future Work **55**
References **56**
Appendix: PHP Source Codes for the MPRV Algorithm **61**

List of Tables

Table 4.1**:** Processes in the ready queue at λ_0 35

Table 4.2: Process sorted in ascending order and TQ is set to the median at λ_1 35

Table 4.3: Values of ETAT and EWT computed at λ_2 36

Table 4.4: Process Snapshot of 10 randomly generated processes 37

Table 4.5: Processes sorted in increasing order of burst time, with their respective EWTs and ETATs 46

Table 4.6: Average waiting time Results obtained 49

Table 4.7: Average Turnaround time Results obtained 50

Table 4.8: Average penalty ratio results obtained 51

Table 4.9: Penalty ratio variance results obtained 52

Table 4.10: Context switching results obtained 53

List of Figures

Figure 2.1: Layers and views of a computer system 6
Figure 2.2: Process Control Block 8
Figure 2.3: Diagram of process state 9
Figure 2.4: Three-level scheduling 12
Figure 2.5: Mid-term scheduler 13
Figure 3.1: The algorithm architecture 28
Figure 3.2: MPRV Flowchart 32
Figure 4.1: RR Gantt chart 38
Figure 4.2: DQRRR Gantt chart 39
Figure 4.3: IRR Gantt chart 41
Figure 4.4: NIRR Gantt Chart 43
Figure 4.5: MPRV Gantt chart 46
Figure 4.6: Graphical Representation of the Average waiting time 49
Figure 4.7: Graphical Representation of the Average turnaround time 50
Figure 4.8: Graphical Representation of the Average penalty ratio 51
Figure 4.9: Graphical Representation of the Penalty ratio variance 52
Figure 4.10: Graphical Representation of the context switching values 53

List of Abbreviations

AMMRR: Average Mid Max Round Robin

ATAT: Average Turn-Around Time

AWT: Average Waiting Time

APR: Average Penalty Ratio

BT: Burst Time

CBT: Combinational Burst Time

CPU: Central Processing Unit

CS: Context Switch

DQRRR: Dynamic Quantum Re-adjusted Round Robin

ETAT: Expected Turnaround Time

EWT: Expected Waiting Time

FCFS: First Come First Served

I/O: Input Output

LJF: Longest Job First

NIRR: New Improved Round Robin

IRR: Improved Round Robin

OS: Operating System

OTQ: Optimal Time Quantum

PCB: Process Control Block

PRRDTQ: Precedence Based Round Robin with Dynamic Time Quantum

PRV: Penalty Ratio Variance

RR: Round Robin

SJF: Shortest Job First

SJN: Shortest Job Next

SPT: Shortest Processing Time

TQ: Time Quantum

CHAPTER ONE

GENERAL INTRODUCTION

1.1 BACKGROUND

One of the main functions of the operating system is scheduling (Babu *et al.,* 2012). By switching the CPU among processes, the operating system can make the computer more productive (Silberschatz *et al*, 2005). Hence, it has become an interesting and important field of research in operating systems.

By way of definition, CPU scheduling can be defined as the art of determining which process to run when there are multiple runnable processes, (Chaudhary *et al.*,2012). It also refers to the way processes are assigned to run on the available CPU (Curtis *et al.,* 2012).CPU scheduling therefore involves selecting from the processes in memory that are ready to execute, and allocating the CPU to one of them. A process is a program in execution.

CPU scheduling is not planning since there is no optimal solution; rather it is about balancing goals and making difficult tradeoffs. A new scheduling algorithm either places emphasis on a different goal or provides a simpler way to achieve the same goals as its predecessors. The only bad scheduling policy is one that trades something for nothing or de-emphasizes one goal without an improvement in another (Meehean, 2011). There exist various CPU scheduling algorithms, each one with its advantages and disadvantages (Lenka and Ranjan, 2012). In order to measure and compare the efficiency of a scheduling algorithm some criteria were suggested (Das *et al*., 2012). Which criteria are used for comparison can make a substantial difference in which algorithm is judged to be the best.

This research focuses on minimizing the variance of penalty ratio of the processes. The penalty ratio of a process is the elapsed time from running the process divided by the sum of the CPU and I/O demands of the job. It is also called the response ratio and is a factor of the turnaround time (Sharma *et al.*, 2012). Another area of concentration in this research will be with making sure that every process makes progress, thereby preventing starvation. Starvation is a condition in which process are made to wait indefinitely for allocation to required resources.

Expected values for turnaround time and waiting time which may or may not be met up with after the processes have been executed will be generated and worked with. However, in the long run, the variance of the penalty ratio values will be minimized. Comparison of results will be done with the following round robin scheduling algorithms: classical Round Robin (RR), Dynamic Quantum Re-adjusted Round Robin (DQRRR), Improved Round Robin (IRR) and New Improved Round Robin (NIRR) algorithms. This is so because the proposed algorithm is a modification of the round robin algorithm.

The results to be compared will be popular CPU scheduling criteria, namely: average waiting time, average turnaround time and context switching. However, two other scheduling uncommon scheduling criteria will also be introduced and used as a basis of comparison. These are: average penalty ratio and penalty ratio variance.

1.2 RESEARCH QUESTIONS

a. How much does the proposed algorithm gain on the minimized penalty ratio variance?
b. What are the trade-offs of the proposed algorithm?

1.3 MOTIVATION OF THE STUDY

Processes that are run using the round robin algorithm have high turnaround time (Abdullahi, 2012), which implies a high variance in penalty ratio values. However, if this variance is reduced, the quality of service experienced by the user can be maximized (Ye *et al, 2005*).

1.4 AIM OF THE STUDY

The aim of the research is to design and implement an algorithm that minimizes the variance of the penalty ratio of the processes in the computer system.

1.5 OBJECTIVES OF THE STUDY

The objectives of the research are to:

a. Modify the round robin scheduling algorithm such that it minimizes the variance of the penalty ratio.
b. Implement the Minimized Penalty Ratio Variance algorithm.
c. Evaluate other modifications to the round robin algorithm, namely RR, IRR, NIRR, DQRRR.

1.6 METHODOLOGY

The approaches that will be applied in achieving our aim and objectives in this research are to:

a. Make extensive study of scheduling algorithms.
b. Develop the proposed algorithm.
c. Implement the proposed algorithm using the PHP programming language.

d. Evaluate the proposed algorithm by comparing it with already existing CPU scheduling algorithms.
e. Summarize and recommend future research.

1.7 SCOPE

The work is limited to the round robin CPU scheduling algorithm. The proposed algorithm is a modification of the classical round robin algorithm.

1.8 SIGNIFICANT CONTRIBUTIONS

The research hopes to make a meaningful contribution in the problem of high turnaround times generated by the Round Robin algorithm. In minimizing the penalty ratio variance, the Round Robin algorithm uses the penalty ratio of each process to determine their turnaround time, thereby achieving service timeliness.

1.9 ORGANIZATION OF THESIS

In chapter two, existing literature and researches conducted in the field of process scheduling are discussed.

In chapter three, the methodologies adopted to design the proposed framework for the Minimized Penalty Ratio Variance Algorithm are discussed.

In chapter four, the implementation and testing of the proposed algorithm is presented.

In chapter five, the summary of the work and proposed areas for future work are presented.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION TO OPERATING SYSTEMS

Computer software can be divided into roughly two categories: system programs: which manage the operations of the computer itself, and application programs; which perform the actual work the user wants. The most fundamental system program is the operating system, whose job is to control all the computers' resources and provide a base upon which the application programs can be written (Tanenbaum and Woodhull, 2006). Put in another way, the operating system is a program that controls the execution of application programs and acts as interface between applications and the computer hardware. This is illustrated in Fig 2.1.

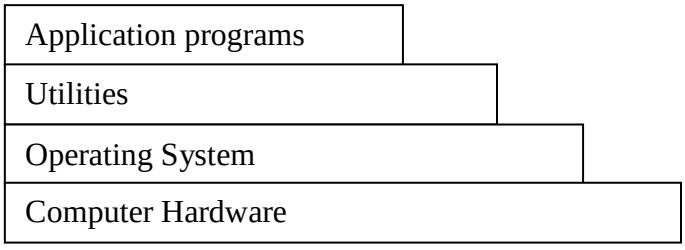


Fig 2.1 Layers and Views Of a Computer System (Tanenbaum and Woodhull, 2006)

The operating system is responsible for managing processes, memory and storage, as well as protection and security. It has three main objectives:

a. Convenience: it makes the computer more convenient to use by acting as a user/computer interface.
b. Efficiency: it allows the computer system resources to be used in an efficient manner.

c. Ability to evolve: it should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without interfering with service (Stallings, 2012).

Operating systems have evolved from generation to generation of computers:

a. Serial Processing: involved users having access to the computer in series.
b. Simple batch systems: jobs are batched together sequentially and run on the computing system.
c. Multiprogrammed batch systems: the processor switches from one program to the next, especially when the previous program is waiting for I/O. The systems became necessary since I/O devices were much slower when compared to the processors, hence resulting in a great lapse in the operating system because the processor was often kept idle (Tanenbaum and Woodhull, 2006).

2.2 PROCESSES

Simply put, the term ‘process’ refers to an executing set of instructions (Harris, 2002). It can also be thought of as a program in execution (Silberschatz *et al*, 2005). Systems consist of a collection of processes. As has been previously said, the operating system manages processes among other things (Tevfik, 2013).

Associated with a process are its code, its data, the operating system resources allocated to it, and one or more threads of execution. A thread is a flow of execution through the process code with its own program counter, system registers and stacks. It is sometimes called a light weight process.

Each process is represented in the operating system by a process control block (PCB) as shown in Fig 2.2.

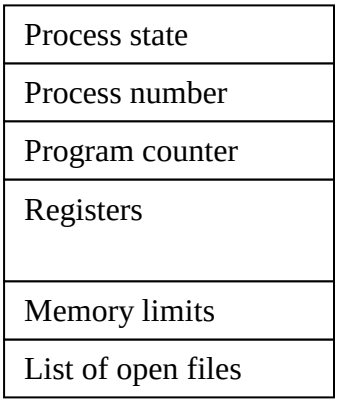


Fig 2.2 Process Control Block (Silberchatz *et al*, 2005)

As seen in the Fig 2.2, the process control block contains the:

a. Process state: This may be new, ready, running, waiting, halted, etc.
b. Process Counter: This indicates the address of the next instruction to be executed for the process.
c. CPU registers: These include accumulators; index registers, stack pointers, general purpose registers plus any condition-code information.
d. CPU scheduling information: this includes the process priority, pointers to scheduling queues and any other scheduling parameters.
e. Memory management information: this includes such information as the value of the base and the limit registers, the page tables or segment tables.
f. Account information: this includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, etc.
g. I/O status information: This includes the list of I/O devices allocated to the processes.

Processes in a short-term scheduler can pass from state to state as illustrated in Fig 2.3. It is called the diagram of process state.

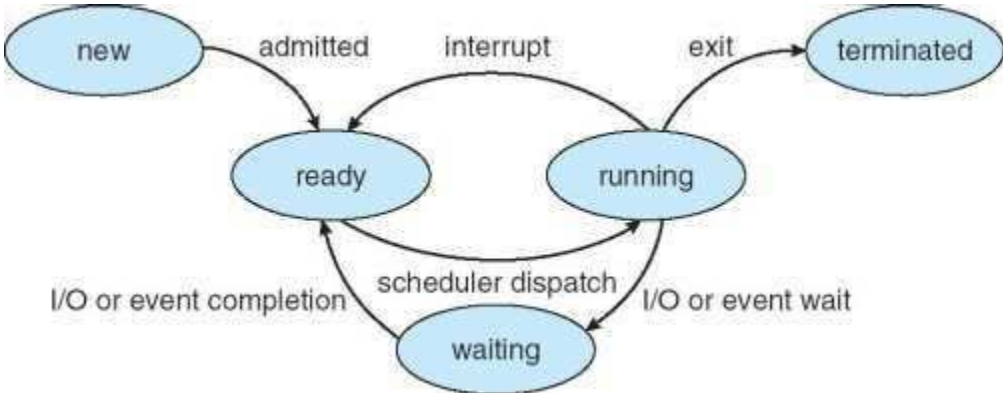


Fig 2.3 Diagram of Process State (Siberschatz *et al*, 2005)

The states are defined as follows:

a. New: Here, the process is created.

b. Ready: After the process has been admitted, it enters a ready queue either according to arrival time or in conformity to a particular CPU scheduling algorithm, waiting to be assigned to the processor. A process can also be in the ready state after an interrupt occurs on it while it is running, or after an event has completed when it is in the waiting state.

c. Running: Instructions are executed on the process. The process is usually executed based on a particular CPU scheduling algorithm after the dispatcher has selected it to run.

d. Waiting: The process is waiting for some event to occur (such as I/O completion or reception of a signal).

e. Terminated: The process exits after execution.

CPU scheduling decisions may take place when a process switches from running to waiting state, running to ready state, and waiting to ready state or terminates.

2.3 SCHEDULING

By way of definition, scheduling is the method or actions by which threads, processes or data flows are given access to system resources (e.g. process time, CPU, communication, bandwidth) which is usually done to balance a system and achieve best quality of service (Ernst *et al.*, 2007; Mostafa *et al.*, 2010). Scheduling also refers to the series of actions performed on a system to distribute processes to resources using the long, short and mid-term scheduler (Abdullahi, 2012).

An operating system must allocate computer resources among potentially competing requirements of multiple processes. The CPU is one of the primary resources, and therefore scheduling is central to the operating system (Yadav *et al.*,2012). The means of allocation of resources is referred to as scheduling (Haque, 2012). In a multiprogramming system, multiple processes exist concurrently in memory, therefore scheduling is important in such an environment (Stallings, 2012). This implies that CPU schedulers provide an illusion of multiple virtual CPUs to applications (or processes); each application appears to have its own CPU (Meehean, 2011).

One of the aims of CPU scheduling, then, is to optimally divide CPU resources amongst competing tasks. Optimality can vary between applications. Therefore, each CPU scheduler needs a system-specific policy that defines how to share the processor. This policy encapsulates the broad scheduling goals of the system, and reflects the system’s expectations regarding its workloads.

Scheduling policy is a balancing act between competing goals. Modern scheduling policies make tradeoffs between three primary goals among others:

a. Fairness: This concerns how CPU cycles are divided over some time scale- which is referred to as its CPU allocation. A policy's fairness can be measured by how closely and at what time scale it matches a desired allocation.
b. Latency: This is the measure of how quickly a task is scheduled after it has become eligible. In other words, it is how long a task must wait before it is given control of the CPU.
c. Progress: This measures the work a task can accomplish in a given time period. In the extreme case, called starvation, a task may make no progress at all. (Meehean, 2011)

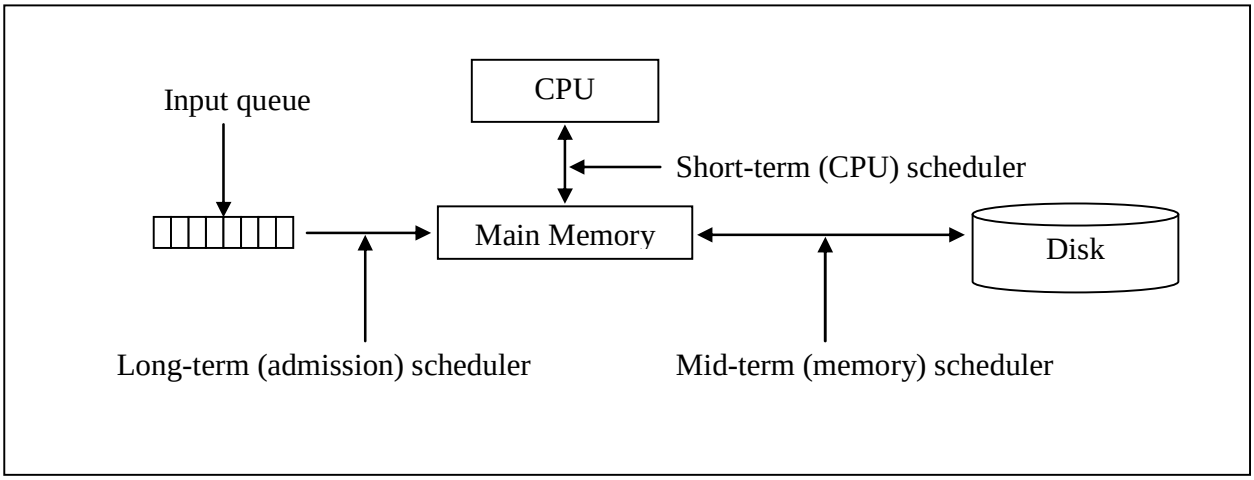


Fig 2.4 Three-Level Scheduling (Tanenbaum and Woodhull, 2006)

Fig 2.4 illustrates Long-, Mid- and Short-term scheduling.

2.3.1 Long-term or admission scheduling

As jobs arrive at the system, they are initially placed in the input queue stored on the disk. As seen in Fig 2.4, the long-term or admission scheduler decides which jobs to admit into the system. The others are kept in the input queue until they are selected (Tanenbaum and Woodhull, 2006). The long-term scheduler also dictates to the system its degree of concurrency and how the processes will be split (Abdullahi, 2012).

Processes being scheduled here can exist in the held state. When a process is in this state, it has been created but will not be considered for loading into memory or for execution. Typically, only new processes can exist in this state (Harris, 2002).

2.3.2 Mid-term or memory scheduling

From Fig 2.4, it can be seen that this second level of scheduling decides which process should be kept in memory and which one should be kept on disk. It can temporarily remove processes from main memory to the secondary memory (swap out) and vice versa (swap in) as shown in Fig 2.5. This scheduler reviews its decisions on allowing processes on disk to get some service frequently.

The mid-term scheduler optimizes system performance by carefully deciding how many processes it wants in memory and what kind of processes should be in memory at that time. Decision criteria include the following:

a. How long has it been since the process was swapped in or out?
b. How much CPU time has the process had recently?
c. How big is the process?

d. How important is the process? (Tanenbaum and Woodhull, 2006)

In modern operating systems, the mid-term scheduler performs the job of the long-term scheduler by the use of virtual address space technology of the memory (Abdullahi, 2012).

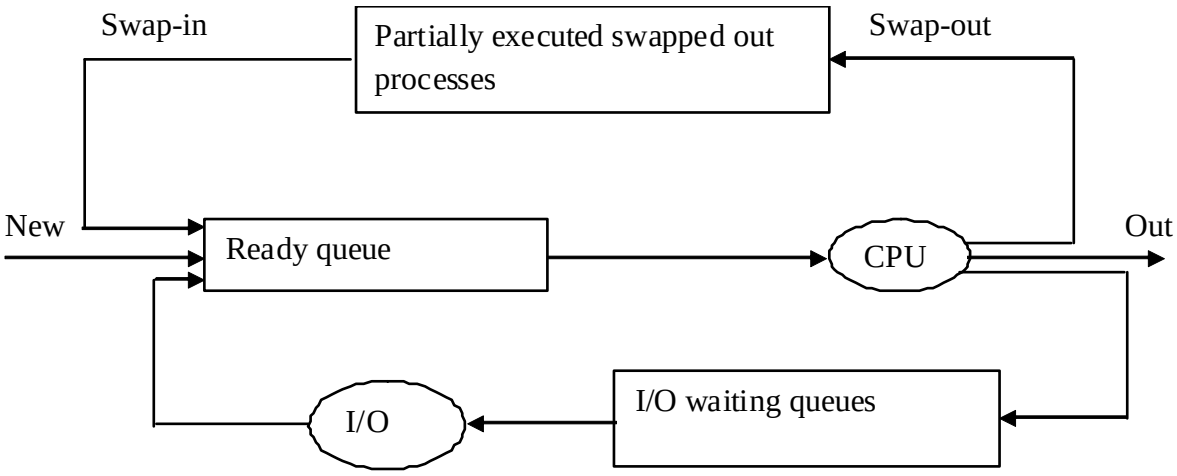


Fig 2.5 Mid-Term Scheduler (Abdulllahi, 2012)

Processes being scheduled by the mid-term scheduler can exist in two states:

a. Swapped-blocked: Here, the process is waiting for some event to occur and has been swapped out to secondary storage.
b. Swapped-ready: The process is ready to run but is swapped out to secondary storage; it may not be allocated to the CPU since it is not main memory (Harris, 2002).

2.3.3 Short-term or CPU scheduling

This is the third level of scheduling. It involves actually picking one of the ready processes in memory to run next by the CPU. The short-term scheduler is invoked whenever an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt currently running process to favour another. Such events include clock interrupts, I/O

interrupts, operating system calls, signals, etc (Stallings, 2012). From Fig 2.3 it is obvious that in short-term scheduling, processes can exist in new, ready, running, waiting and terminated states. It works with any suitable algorithm.

2.3.4 Performance measures for scheduling algorithms (Scheduling criteria)

A. User's point of view

a. Turnaround time: This is the interval between the time of submission of a process and the time of completion of the process. That implies it is the sum of the times the process waited to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.

b. Response Time: This is the measure of time from the submission of a request until the time its first response is produced. Shorter system response time may lead to higher productivity (Schneiderman, 1984).

c. Meeting deadlines: This is the ability for the scheduler to meet predefined deadlines of the processes. It is only required when the minimal execution time of an application can be accurately predicted.

d. Predictability: This is a lack of variance in certain measures of performance in a system, such as its turnaround time, waiting time or response time.

B. System's point of view

a. Throughput: This is the measure of the work completed per unit time.

b. Waiting time: This is the sum of the periods the process spent waiting in the ready queue.

c. User Processor Utilization: This is the percentage of unit time during which the CPU is running user processes.

d. Overall Processor Utilization: This is the time percentage during which the CPU is busy.

e. Resource Utilization Balance: This extends the idea of processor utilization to take into account all system resources. A good scheduler should try to keep all the hardware resources in use at any time.

f. Penalty Ratio: This is the elapsed time divided by the sum of the CPU and I/O demands of the job. It is a better measure of how well the scheduler is doing since it is a factor of the turnaround time. If the penalty ratio is low, it means the turnaround time is better and vice versa. If its variance is low, then by implication, the turnaround time is stable and more predictable. It is also called the response ratio.

Penalty Ratio (PR) = Turnaround time (TAT) / Burst time (BT)

C. Interaction point of view

a. Fairness: This implies that no process should be left to starve waiting for the CPU for the CPU time.

b. Policy Enforcement: Within the limits of a fair scheduling policy, preference should be given to high priority processes, when priorities are defined (Sharma *et al*, 2012).

2.4 SCHEDULING ALGORITHMS IN OPERATING SYSTEMS

As has been previously said, CPU scheduling is the basis for multiprogrammed operating systems. The CPU scheduler works with any suitable algorithm. Processes require a scheduling algorithm to decide on the mode and nature of the execution process (Bolesaw, 1990).

2.4.1 First-Come-First-Served (FCFS)

This is the simplest scheduling policy and also known as the first-in-first-out (FIFO) (Stallings, 2012). This implies that the algorithm follows a strict queuing scheme: the first process/job that arrives is the first to be scheduled; therefore the first job in the queue will be the first to leave. The algorithm is easily managed with a FIFO queue (Oyetunji and Oluleye, 2009).

A. Properties of FCFS

a. Selection function is based on the process with maximum waiting time.

b. Decision mode is non-preemptive.

c. Minimal context switching overhead.

d. Response time may be high.

e. Throughput maybe low especially when longer jobs are executed first.

f. Short processes and I/O bound processes are penalized, since it long jobs are favoured.

g. No starvation of processes.

B. Advantages of FCFS

a. Fairness: Each process is treated fairly depending on the time of arrival.

b. Less costly to implement.

c. No pre-emption.

d. Better for shorter jobs.

C. Disadvantages/problems of FCFS

a. High turnaround time.

b. Starvation of shorter jobs when longer jobs exist.

c. Un-attainable due dates.

d. Reduces CPU and I/O device utilization.

e. Could result in convoy problem. (Abdullahi, 2012).

2.4.2 Shortest-Job-First Scheduling (SJF)

In this case, the scheduler always chooses the process that has the shortest expected remaining processing time. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie (Silberschatz *et al.*, 2005). The SJF may be implemented as either preemptive or non-preemptive (Oyetunji and Oluleye, 2009). SJF with preemption forcibly takes a process out of the CPU when a job with a lesser burst time arrives. SJF without preemption on the other hand allows the process being executed by the CPU to run fully before handing control over to the ready process with the shortest burst time. The SJF algorithm favouritism of shorter processes over longer ones is an overhead when compared to FCFS (Bandarupalli *et al.*, 2012). It is the most popular scheduling algorithm (Jaiswal, 2012).

A. Properties of SJF

a. Selection function is based on the process with the shortest service time.

b. Decision mode can either be preemptive or non-preemptive.

c. Throughput is high.

d. Response time is good, especially for short processes.

e. High context switch overhead.

f. Long processes are penalized, and therefore starved.

B. Advantages of SJF

a. Maximum throughput.

b. Low waiting time compared to FCFS.

c. Response time is better as jobs are treated in a faster manner.

d. Maximum utilization of CPU.

C. Disadvantages/problems of SJF

a. Difficult in checking job pool every time for the next schedule.

b. Next burst-time is usually predicted by the OS using the past burst-time.

c. Interrupts running process.

d. Could lead to starving longer jobs.

e. High cost of implementation because of the run-time estimation. It is not a widely used scheduling algorithm.

f. Selfish and unrealistic (Abdullahi, 2012).

2.4.3 Round-Robin scheduling (RR)

The round-robin scheduling can be referred to as preemptive FCFS. This is because fixed time slices called ‘time quantum’ are assigned to the processes as they are scheduled. The algorithm gives a better response time, but worse turnaround and waiting times because of the static time quantum (Srivastav *et al*, 2012). If the time quantum is too small, it results in overhead due to context switching. However, if it is too large, then the processes would be executed just like the normal FCFS.

A. Properties of RR

a. Every process has the opportunity to run for its designated time slice.

b. Decision mode is preemptive; processes are preempted when they exhaust their time quantum.

c. It provides good response time for short processes.

d. All processes are treated fairly, therefore there is no starvation.

B. Advantages of RR

a. Easy to implement

b. Each process gets equal share of the CPU.

c. CPU utilization is almost 100% as the CPU never gets idle.

C. Disadvantages/problem of RR

a. Longer jobs keeps switching when Quantum is low (Abdullahi, 2012).

2.4.4 Priority scheduling

In the priority scheduling algorithm the CPU is allocated to processes according to the priority already assigned to each process. Priority can be determined by the user or by the system. Priority assignment can be fixed or dynamic and the approach adopted can be preemptive or non-preemptive (Saxena and Agarwal, 2012). Processes with higher priorities are executed before those with lower priorities. Where multiple processes have the same priority, the FCFS approach is used to break the tie.

A. Properties of Priority Scheduling

a. Overhead is not minimal.

b. All evaluation metrics depend on the priority order.

c. High priority processes have smaller waiting time

d. High priority processes have smaller response time.

e. Deadlines are not met by assigning deadlines to jobs with lower priority.

f. Starvation is likely to occur to processes with lower priority.

B. Advantages of Priority Scheduling

a. Throughput is easily predicted and enhanced as priority are set.

b. High turnaround time.

c. High response time.

C. Disadvantages/problems of Priority Scheduling

a. Starvation as a result of lower priority score.

b. Costly as involves context switching between processes of lower priority scores (Abdullahi, 2012).

2.4.5 Highest Response Ratio Next (HRRN)

The highest response ratio next algorithm was proposed by Brinch Hansen to avoid the limitations of the shortest job first algorithm (Pattanayak *et al.*, 2012). It is a non-preemptive algorithm that selects the process that has the highest response ratio, as calculated from the equation:

$$RR = (w + s)/\, s \qquad (2.1)$$

Where RR = Response ratio

w = time since arrival

s = service time of the process

When the process just arrives to the system, it has a response ratio of 1. The response ratio keeps increasing as the process is waiting in the system. The process with a short service time generally has its response ratio increasing faster than that with a longer service time.

A. Properties of HRRN

a. The ready queue is maintained in the order of the response ratio (Barade and Khande, 2010).
b. The priority of the job is dependent on its estimated run time, and also the amount of time it has spent waiting in the ready queue (Pattanayak *et al.,* 2012).

B. Advantages of HRRN

a. It minimizes the average response ratio of each process (Niu, 2003)
b. Long jobs eventually get a turn through aging, even though short jobs are favoured.

C. Disadvantages of HRRN

a. It fails in responsiveness due to its non-preemptive mode (Barade and Khande, 2010)

2.4 RELATED WORK

Mohanty et al., (2010) adopted an approach of re-adjusting the time quantum with every round robin cycle. This re-adjustment was achieved by finding the median of all the remaining CPU bursts of the processes, and setting the time quantum to this value for the new cycle. When the DQRRR algorithm was compared to the round robin algorithm, it gave better average turnaround and waiting times.

Mohanty *et al.* (2011) introduced the intelligent time slicing for the round robin algorithm which depends on three things: priority, average CPU burst and context switch avoidance time. The intelligent time slice of individual processes is computed by summing up the priority component, shortness component, context switch component, and original time component of the individual processes. These parameters are arrived at by using various mathematical computations. Based on the intelligent time slice arrived at for each process, they are assigned to the CPU. The algorithm was found to be effective for real time systems.

Behera *et al.*, (2012) proposed the “Round Robin with Highest Response Ratio Next for Soft Real Time Systems”. In this algorithm the time quantum is set to the mean of the burst times of the processes. The ready queue is filled according to the arrival time of the processes. Eligibility for a particular process to run is determined by computing the response time of each process and is calculated using the following formula: Waiting time + Burst time / Burst time. The process with the highest value is assigned to the CPU. After each process runs or after a new process arrives, the waiting times of all the processes in the run queue are updated and used to order the processes in each RR cycle. When the algorithm was compared to the DQRRR it was found to have better performance in terms of average waiting time, turnaround time and context switches.

Nayak *et al.*, (2012) applied another approach to the round robin algorithm. They modified the algorithm and called their work “Improved Round Robin Scheduling using Dynamic Time Quantum”. The research introduced the “optimal time quantum (OQT)” which is computed as the mean of the highest burst time and the median of all the CPU bursts of the processes. The OQT is assigned to each process and is recalculated taking the remaining burst time into account after each cycle, and the procedure is continued until the ready queue is empty. Their goal was that the OQT made the algorithm give better average turnaround and waiting times, minimized context switches, especially when compared with the round robin algorithm.

Dawood (2012) proposed an algorithm in which the time quantum is computed by finding the sum of the maximum and minimum CPU bursts of the processes in the ready queue, and multiplying the sum by 80%. This multiplication is based on the rule of thumb that 80% of the CPU bursts should be shorter than the time quantum. The processes are assigned to the CPU based on the computed time quantum, and the remaining burst times are used to compute a new time quantum for the next round. The algorithm was found to win on average waiting time,

average turnaround time and context switching when compared to the round robin and some other variations of it.

Abdullahi (2012) introduced the longest job first algorithm, which performs exactly opposite of the shortest job first algorithm because it runs longer jobs first, thereby starving shorter jobs. However, the starvation problem was solved by coming up with the combinational burst time (CBT) model that combines shorter jobs based on certain conditions, and runs them according to their burst time lengths. The algorithm not only solved the starvation problem in the longest job first algorithm, but also minimized context switching when compared with the classical FCFS, SJF, Priority, RR and LJF algorithms.

Banerjee et al., (2012) worked on the "Performance Evaluation of a New Proposed Average Mid Max Round Robin (AMMRR) Scheduling Algorithm with Round Robin Scheduling Algorithm". In their work, the processes in the run queue are sorted according the burst time of the processes and the time quantum is computed by finding the average of the sum of the mid and max values. The mid value is computed by dividing the sum of the min and max values by two. The TQ is modified in every RR cycle such that for every cycle there is a new time quantum and the ready queue is constantly reordered with every new process that arrives. When the algorithm was compared with the round robin algorithm, it had better values of the average turnaround and waiting times, and context switches.

Behera and Swain (2012) also worked on "A New Proposed Precedence Based Round Robin with Dynamic Time Quantum (PRRDTQ) Scheduling Algorithm For Soft Real Time Systems". Here, a new factor known as the RR algorithm known as the Precedence factor is introduced by combining the basic factors of the process, which is priority and burst time. The priority factor is computed by summing up the products of the priority and priority ratio and the burst time and

burst ratio respectively. The priority factor is found by dividing the priority of the process and the total number of processes. The burst ratio similarly is computed by finding the quotient of the burst time of the processes and the summation of the burst time of the processes. After all these computations are made, precedence is assigned according to their factors, with those that have lesser factors getting higher precedence. Sorting of the processes in the ready queue is then done based on the new precedence and the RR algorithm is applied with dynamic time quantum, which is computed by taking the mean of the remaining burst times. The algorithm had it that processes with shorter bursts and higher priority executed first resulting in better turn around and waiting times when compared with the RR algorithm.

Abdulrahim *et al.* (2014) proposed an algorithm with two queues for the processes: the arrive and request queues. When a process arrives, it goes to the arrive queue and from there to the request queue; from which it is admitted into the CPU. When the first process arrives, it goes straight to the request queue and is run by the CPU to completion. Afterwards, subsequent processes in the queue are sorted in ascending order of bursts in the request queue. The time quantum is set to the median of the burst times of the processes. Then the first process in the request queue is allocated to the CPU for a period of one time quantum. If the remaining burst time of the process is less than half of the time quantum, that process is re-assigned to the CPU, else the next process in the request queue is. When the algorithm was compared to the round robin algorithm, it was found to win on average waiting time, average turnaround time and number of context switches.

CHAPTER THREE

MINIMIZED PENALTY RATIO VARIANCE ALGORITHM FRAMEWORK

3.1 INTRODUCTION

Process scheduling is an important measure of the efficiency and effectiveness of an operating system (Neetu and Garg, 2012), especially since part of its responsibility is to control allocation of system resources to processes; of which the CPU is a part.

In the research, an algorithm that minimizes the variance of the penalty ratio of the processes is proposed. This is especially important since the penalty ratio is individualistic for processes, and minimizing the variance of this value brings about service timeliness in the system. In other words, from the user's point of view, a minimized variance in the penalty ratio- which results in a more stable turnaround time- maybe more desirable than a minimized mean penalty ratio value which may vary greatly from user to user. Using the mean values of various scheduling criteria as a measure of system performance may not be the ultimate since these mean values in the real sense are not experienced by any particular user. Instead, they are experienced by a "virtual" user whose demand distribution is the same as the demand distribution presented by the whole user population (Jiang and Vaidya, 1998). Therefore, these mean vales of the performance criteria may not be optimal for a particular process/job in the system. It may be more desirable to study how these performance criteria of a scheduling algorithm fare on a process level than how the general performance of the algorithm is- which is what is achieved when the mean values are computed and evaluated.

3.2 APPROACH OF THE MODEL

For this research, the ratio of the sum of the burst time of all the processes in the system and the service time of the process to the largest burst is computed. This will be called the target penalty ratio. Every process in the system will be run in such a way as to make their respective penalty ratio values either to be the same as this target value or to fall within a low range from this target value. To say it in mathematical terms, the variance of the penalty ratio becomes smaller. The algorithm takes some cue from the existing classical scheduling algorithms; such as time quantum from the round robin algorithm, sorting of processes according to burst time from the shortest job first algorithm, and running jobs with the same burst time according to which precedes the other from the first come, first served algorithm. The time quantum of the MPRV algorithm will be set to the median of the service time of all the processes. This ensures that up to half of the processes are run in the first cycle, and therefore minimizes the gruesome effect context switching especially since it is a preemptive scheduling algorithm.

Long processes are penalized in almost every CPU scheduling algorithm. Therefore, this problem of starvation of long processes is sought to be solved by using them to “fill in” for the expected waiting times of the shorter processes. This implies that longer jobs make progress and shorter jobs with bursts lesser than the time quantum execute completely once they have been given control of the CPU. Fig 4.1 shows the architecture of the algorithm.

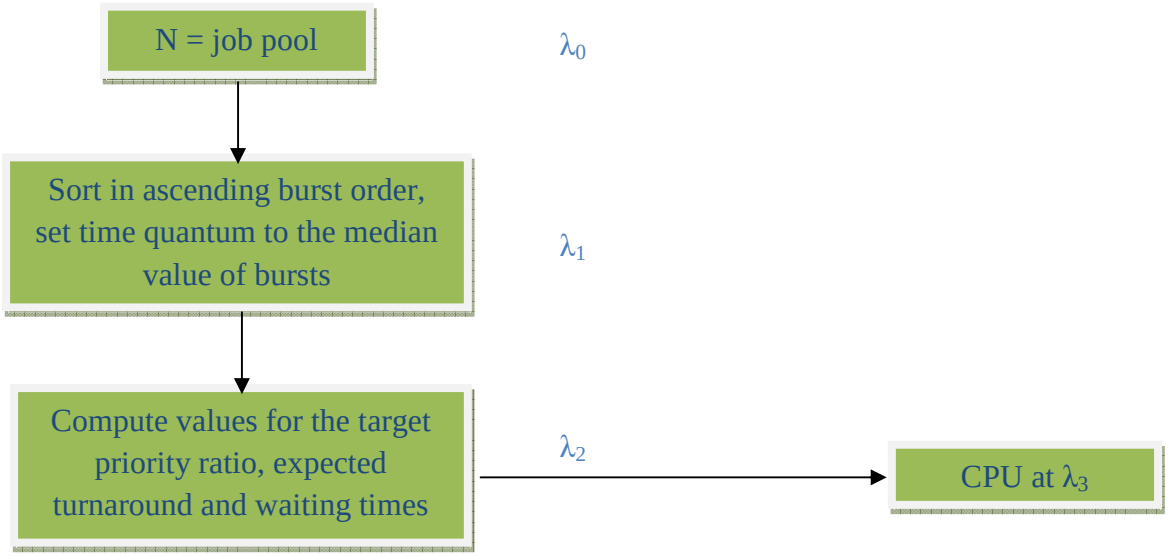


Fig 3.1 Architecture of the Algorithm

3.3 THE MINIMIZED PENALTY RATIO VARIANCE ALGORITHM (MPRV)

Consider a job pool N as a set containing various processes X with distinct burst times δ for each. There also exists λ as a phase of process execution.

$N = \{X_1, X_2, X_3, \ldots, X_n\}$ with burst-times $\delta_1, \delta_2, \delta_3, \ldots, \delta_n$ for every δ_i associated with X_i at λ_0.

The algorithm requires the median value of the service (burst) times so as to set it the time quantum of the algorithm. It is therefore of necessity to order the processes according to their burst times to make it easy to evaluate. This is done at the λ_1 phase of execution.

At λ_2, the Target Penalty Ratio (TPR) of the processes along with the Expected Turnaround Time (ETAT) and Expected Waiting Time (EWT) are computed. They are calculated as follows:

$$TPR = \frac{\sum BT}{LBT} \qquad (3.1)$$

$ETAT(P_i) = ROUND\ (TPR*BT(P_i))$ (3.2)

$EWT(P_i) = ETAT(P_i) - BT(P_i)$ (3.3)

Where

BT : burst time of processes

LBT : longest burst time

$BT(P_i)$: burst time of process P_i

$ETAT(P_i)$: expected turnaround time of process P_i

$EWT(P_i)$: expected waiting time of process P_i

On completion of the phase λ_2, the algorithm then admits processes into CPU.

3.3.1 The proposed model

The order which the algorithm follows is:

Let $X_i\delta_i$ be a process (X) with id number (i) with its burst time (δ_i):

a. Submit processes to pool N at λ_0.
b. Sort all processes according to ascending order of burst times. Set the time quantum (TQ) to the median value of their burst times at λ_1.
c. Determine Target Penalty Ratio for all processes.
d. For each processes $P_1 \leq P_i \leq P_n$, compute the values of Expected Turnaround Time and Expected Waiting Time at λ_2.
e. If $EWT(P_i)$ – Runtime (RT) ≤ 0 run P_i, else run longest remaining burst first (LRBF).

f. Preemption of the LRB is on two conditions:

   i. when RT = TQ ; after which the next LRB runs, or

   ii. when RT = $EWTP_i$; after which P_i runs.

g. Do (e) and (f) while processes are in the ready queue.

3.3.2 MPRV algorithm

a Start

b Sort all processes according to ascending order of burst times.

c Set the time quantum (TQ) to the median value of their burst times.

d Target Penalty Ratio (TPR) = ∑Burst time of processes ($BT(P_i)$) /Longest Burst Time.

e Set Runtime (RT) = 0

f For each processes $P_1 \leq P_i \leq P_n$

i. Expected Turnaround Time ($ETAT(P_i)$) = $BT(P_i)$* TPR

ii. Expected Waiting Time ($EWT(P_i)$) = ETAT - $BT(P_i)$.

g For each Process $P_1 \leq P_i \leq P_n$:if $EWT(P_i)$ - RT ≤ 0 run P_i else run longest remaining burst(LRB) until RT(LRB) = TQ or $EWT(P_i)$ – RT = 0

h If RT(LRB) = TQ, run next LRB, else If $EWT(P_i)$ – RT = 0, run P_i

i While ProcessInReadyQueue > = 1 loop g and h

j Compute Average TAT, WT, PR and variance of PR.

k Stop

The flowchart of the proposed model is as shown in Fig 3.2

3.2.3 FLOWCHART OF THE PROPOSED MODEL

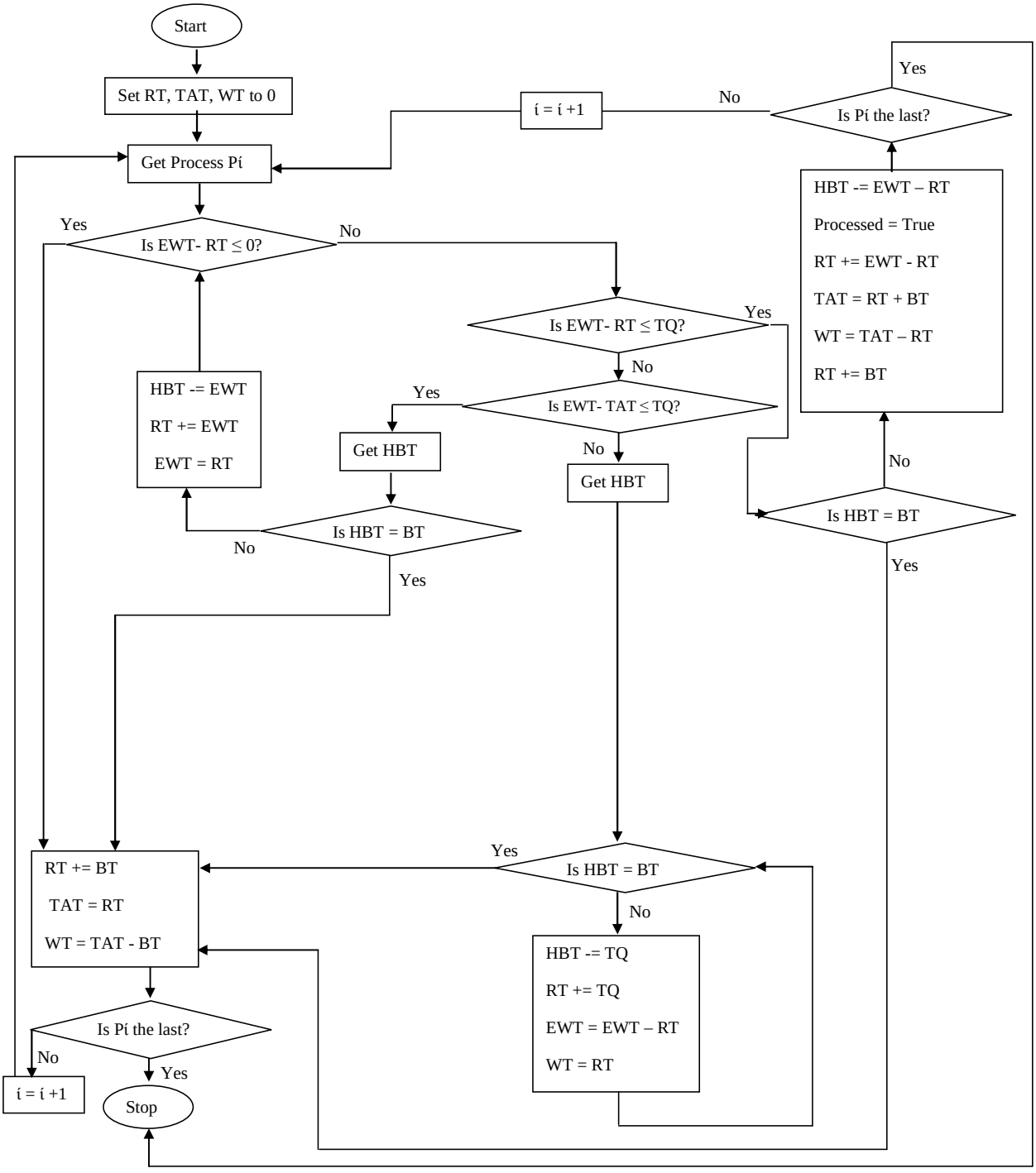


Fig 3.2: MPRV Flowchart

CHAPTER FOUR

IMPLEMENTATION AND TEST OF THE MINIMIZED PENALTY RATIO VARIANCE ALGORITHM

4.1 INTRODUCTION

Different cases of implementation of the MPRV model were tested after the model was designed in PHP to create an interface where processes are entered and the ranges of burst times are specified in order to simulate the mode of operation of the algorithm. Then the sample results are analyzed and compared with the results obtained from the popular existing scheduling algorithms.

4.2 CASE ASSUMPTION

In a multiprocessing environment, processes of different characteristics are submitted to the CPU, this research assumed a system comprising of 10-1200 processes to be assigned the CPU based on the MPRV model. The identification of each process was generated and assigned as a sequence of processes, while burst times were randomly generated.

It was also assumed that the processes did not have any I/O demands; hence only the burst times of the process in the computation of the average penalty ratio and penalty ratio variance was required and used.

4.3 SYSTEM REQUIREMENTS

4.3.1 Notepad ++

Notepad++ is a free source code editor and Notepad replacement that supports several languages. Running in the MS Windows environment, its use is governed by GPL License. Based on the powerful editing component Scintilla, Notepad++ is written in C++ and uses pure Win 32 API and STL which ensures a higher execution speed and smaller program size.

4.3.2 PHP Language

PHP stands for the acronym PHP: Hypertext Preprocessor. It started out as a set of tools for doing web-related tasks, and did nothing more than track visits to a particular web page. It was then expanded to interact with databases, as well as provide a tool set for online guest books and HTML form processing. Its ability to easily interact with databases such as MySQL remains one of its most attractive features (Brett, 2012). It is a loosely-typed object oriented language.

4.3.3 Simulation System specifications

a. Processor speed: 2.10 GHz

b. Installed Memory (RAM): 4.00GB

c. System type: 64-bit Operating system

d. Hard disk size: 219GB

4.4 SIMULATION FRAMEWORK

The frame work will present different cases of the generated processes in chunks and groups for evaluation and comparisons of different results obtained. Each case shall present the processes involved, and compute the waiting time, turnaround time and their respective averages. Additionally, the variance of the penalty ratio values will also be computed. All these would be considered as the performance metrics of the algorithm. By following all steps of the proposed MPRV framework in Chapter Three, results will be collated and compared to the round robin algorithms.

4.4.1 Processes Case 1

From the results of the generated processes, 10 processes are presented in Table 4.1:

Table 4.1: Processes in the Ready Queue at λ_0

P1 = 61	P2 = 48	P3 = 8	P4 = 56	P5 = 64
P6 = 40	P7 = 21	P8 = 15	P9 = 76	P10 = 88

In Table 4.2, the generated processes are sorted according to ascending order of their respective burst times.

Table 4.2: Processes Sorted in Ascending Order and TQ is Set to Median value at λ_1

P3 = 8	P8 = 15	P7 = 21	P6 = 40	P2 = 48
P4 = 56	P1 = 61	P5 = 64	P9 = 76	P5= 88

TQ = median = 52

At λ_2, the target penalty ratio which is the ratio of the sum of the burst times to the largest burst time is computed. That is,

Target Penalty Ratio (TPR) = $\sum$BT / LBT

= 477 / 88 = 5.42

Computations are done for the expected turnaround time (ETAT) and expected waiting time (EWT) for each process. ETAT was done by finding the product of the TPR and BTP_i, while EWT was done by finding the arithmetic difference between ETAT and BTP_i, as explained in the previous chapter. The resulting values are shown in table 4.3.

Table 4.3: Values for ETAT and EWT computed at λ_2

Process	Burst time	Expected Turnaround Time (EWT)	Expected Waiting time (EWT)
P1	5	43	35
P9	8	81	66
P4	13	114	93
P3	32	217	177
P10	42	260	212
P2	50	304	248
P8	58	331	270
P7	60	347	283
P6	84	412	336
P5	96	477	389

The criterion for the eligibility of a process to be admitted into the CPU at λ_3 is that EWT – RT $\geq$ 0.

4.5 EXPERIMENT CASE ALONGSIDE ROUND ROBIN ALGORITHMS

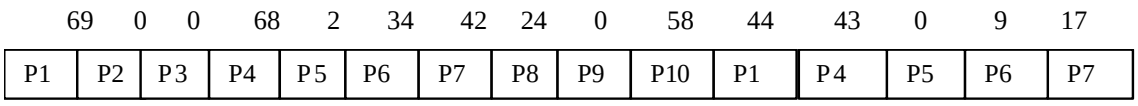
To experiment the arguments of this research, 10 processes shall be randomly generated, all of which shall have distinct identification and all arriving at time zero. The 10 generated processes are shown in Table 4.4. Thereafter, the evaluation shall be tested alongside the Round Robin (RR) category of algorithms namely; classical Round Robin (RR), Dynamic Quantum Re-adjusted Round Robin (DQRRR), Improved Round Robin (IRR) and New Improved Round Robin (NIRR) algorithms.

Table 4.4: Process Snapshot of 10 Randomly Generated Processes

Process	Burst time
P1	94
P2	9
P3	12
P4	93
P5	27
P6	59
P7	67
P8	49
P9	2
P10	83

4.5.1 Classical Round Robin (RR)

The Round Robin scheduling algorithm provides the mechanism for scheduling in cyclic order using a metric measure referred to as quantum. This is usually specified by the system designer (Abdullahi, 2012). The downside of this algorithm is that if the time quantum is too small, it results in too many context switches, and if it is too large, then the algorithm runs just like FCFS. Fig 4.1 represents a generated Gantt chart for the round robin algorithm with a time quantum of 25.



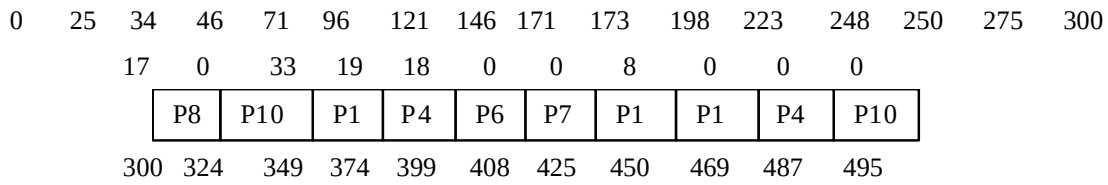


Fig 4.1: RR Gantt Chart

RR:

Turnaround time (TAT) = Overall time a process spent in the system

= Time of Process completion

Average Turnaround time (ATAT) = Sum of each process TAT / Total number of processes

Therefore, TAT in the system will be:

P1: 469, P2: 34, P3: 46, P4: 487, P5: 250, P6: 408, P7: 425, P8: 324, P9: 173, P10: 495.

ATAT = (469 + 34 + 46 + 487 + 250 + 408 + 425 + 324 + 173 + 495) / 10

= 3111 / 10

= 311.1

Waiting time (WT) = TAT – Burst Time

Average Waiting time (AWT) = Sum of all Process' Waiting Times / Total numbers of Processes

P1: (469 - 94) = 375, P2: (34 - 9) = 25, P3: (46 - 12) = 34, P4: (487 - 93) = 394, P5: (250 - 27) = 223, P6: (408 - 59) = 349, P7: (425 - 67) = 358, P8: (324 - 49) = 275, P9: (173 - 2) = 171, P10: (495 - 83) = 412.

AWT = (375 + 25 + 34 + 394 + 223 + 349 + 358 + 275 + 171 + 412) / 10

= 2616 / 10

= 261.6

Penalty Ratio (PR) = Turnaround time / Burst time

Average PR (APR) = Sum of PR of each process / number of processes

Variance of PR (PRV) = (Sum of squares of the PR of each process /number of processes) – APR

Penalty ratio of the respective processes will then be computed thus:

P1: (469 / 94) = 4.99, P2: (34 / 9) = 3.78, P3: (46 / 12) = 3.83, P4: (487 / 93) = 5.24, P5: (250 / 27) = 9.26, P6: (408 / 59) = 6.92, P7: (425 / 67) = 6.34, P8: (324 / 49) = 6.61, P9: (173 / 2) = 86.5, P10: (495 / 83) = 5.96.

$APR = (4.99+ 3.78 + 3.83 + 5.24 + 9.26 + 6.92 + 6.34 + 6.61 + 86.5 + 5.96) / 10$

$= 139.4 / 10$

$= 13.940$

$PRV = ((4.99+ 3.78 + 3.83 + 5.24 + 9.26 + 6.92 + 6.34 + 6.61 + 86.5 + 5.96)^2 / 10) – 13.94^2$

$= (781.6608 / 10) – 194.3236 = 781.6608 – 194.3236$

$= 587.337$

4.5.2 Dynamic Quantum Re-adjusted Round Robin (DQRRR)

Here, the ready queue is sorted in a special way: the process highest remaining burst always comes after that with the least remaining burst. Thereafter, the processes are assigned a time quantum which is the median of the remaining bursts. Figure 4.2 shows the Gantt chart representation.

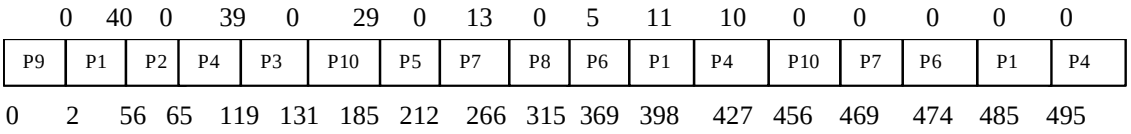


Fig 4.2: DQRRR Gantt Chart

For these set of data, the time quantum values for each cycle are 54, 29, 11 respectively.

DQRRR:

Turnaround time (TAT) = Overall time a process spent in the system

= Time of Process completion

Average Turnaround time (ATAT) = Sum of each process TAT / Total number of processes

Therefore, TAT in the system will be:

P1: 485, P2: 65, P3: 131, P4: 495, P5: 212, P6: 474, P7: 469, P8: 315, P9: 2, P10: 456.

ATAT = (485 + 65 + 131 + 495 + 212 + 474 + 469 + 315 + 2 + 456) / 10

= 3104 / 10

= 310.4

Waiting time (WT) = TAT – Burst Time

Average Waiting time (AWT) = Sum of all Process' Waiting Times / Total numbers of Processes

P1: (485 - 94) = 391, P2: (65 - 9) = 56, P3: (131 - 12) = 119, P4: (495 - 93) = 402, P5: (212 - 27) = 185, P6: (474 - 59) = 415, P7: (469 - 67) = 402, P8: (315 - 49) = 266, P9: (2 - 2) = 0, P10: (456 - 83) = 373.

AWT = (391 + 56 + 119 + 334 + 185 + 415 + 402 + 226 + 0 + 373) / 10

= 2569 / 10

= 256.9

Penalty Ratio (PR) = Turnaround time / Burst time

Average PR (APR) = Sum of PR of each process / number of processes

Variance of PR (PRV) = (Sum of squares of the PR of each process /number of processes) – APR

Penalty ratio of the respective processes will then be computed thus:

P1: (485 / 94) = 5.15, P2: (69 / 9) = 7.22, P3: (131 / 12) = 10.92, P4: (495 / 93) = 5.32, P5: (212 / 27) = 7.85, P6: (474 / 59) = 8.03, P7: (469 / 67) =7, P8: (315 / 49) = 6.43, P9: (2 / 2) = 1, P10: (456 / 83) = 5.49.

APR = (5.15 + 7.22 + 10.92 + 4.59 + 7.85 + 8.03 + 7 + 6.43 + 1 + 5.49) / 10

= 63.68 / 10

= 6.368

PRV = ((5.15 + 7.22 + 10.92 + 4.59 + 7.85 + 8.03 + 7 + 6.43 + 1 + 5.49)^2 / 10) – 6.37^2

= (466.5538 / 10) – 40.5769

= 46.65538 – 40.5769

= 6.078

4.5.3 Improved Round Robin (IRR)

This is a variation of the round robin algorithm. Here, the processes in the ready queue are sorted in ascending order and the time quantum is computed by dividing the sum of the median and highest remaining burst by two in every cycle. Figure 4.3 shows the Gantt chart representation of the IRR algorithm.

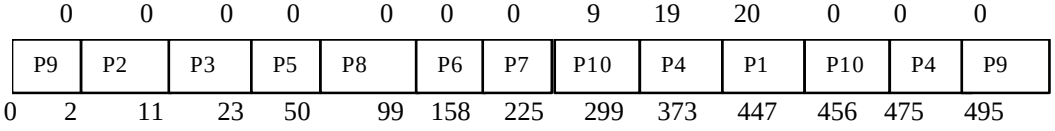


Fig 4.3: IRR Gantt Chart

For these set of data, the time quantum values for each cycle are 74, 20 respectively.

IRR:

Turnaround time (TAT) = Overall time a process spent in the system

= Time of Process completion

Average Turnaround time (ATAT) = Sum of each process TAT / Total number of processes

Therefore, TAT in the system will be:

P1: 495, P2: 11, P3: 23, P4: 373, P5: 50, P6: 158, P7: 225, P8: 99, P9: 2, P10: 299.

ATAT = (495 + 11 + 23 + 373 + 50 + 158 + 225 + 99 + 2 + 299) / 10

= 1735 / 10

= 173.5

Waiting time (WT) = TAT – Burst Time

Average Waiting time (AWT) = Sum of all Process' Waiting Times / Total numbers of Processes

P1: (495 - 94) = 401, P2: (11 - 9) = 2, P3: (23 - 12) = 11, P4: (373 - 93) = 280, P5: (50 - 27) = 23, P6: (158 - 59) = 99, P7: (225 - 67) = 158, P8: (99 - 49) = 50, P9: (2 - 2) = 0, P10: (99 - 83) = 16.

AWT = (401 + 2 + 11 + 280 + 23 + 99 + 158 + 50 + 0 + 16) / 10

= 1040 / 10

= 104.0

Penalty Ratio (PR) = Turnaround time / Burst time

Average PR (APR) = Sum of PR of each process / number of processes

Variance of PR (PRV) = (Sum of squares of the PR of each process /number of processes) – APR

Penalty ratio of the respective processes will then be computed thus:

P1: (495 / 94) = 5.27, P2: (11 / 9) = 1.22, P3: (23 / 12) = 1.92, P4: (373 / 93) = 4.01, P5: (50 / 27) = 1.85, P6: (158 / 59) = 2.68, P7: (225 / 67) =3.36, P8: (99 / 49) = 2.02, P9: (2 / 2) = 1, P10: (99 / 83) = 1.19.

APR = (5.27 + 1.22 + 1.92 + 4.01 + 1.85 + 2.68 + 3.36 + 2.02 + 1 + 1.19) / 10

= 63.68 / 10

= 6.368

$PRV = ((5.15 + 7.22 + 10.92 + 4.59 + 7.85 + 8.03 + 7 + 6.43 + 1 + 5.49)^2 / 10) – 6.37^2$

= (466.5538 / 10) – 40.5769

= 46.65538 – 40.5769

= 6.078

4.5.4 New Improved Round Robin (NIRR)

This algorithm here has it that the first process runs to completion. Subsequent processes are ordered in ascending order of bursts. The time quantum is set to the mean of the burst time of processes. The processes are preempted after they run for on time quantum. If the remaining burst time is less than half of the time quantum, it is reassigned to the CPU, else the next process is run. Figure 4.4 depicts the Gantt chart representation.

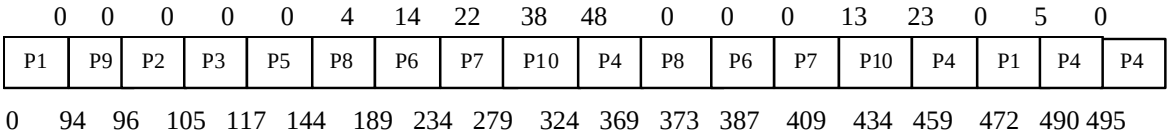


Fig 4.4: NIRR Gantt Chart

For these set of data, the time quantum values for each cycle are 45, 25, 18 respectively.

NIRR:

Turnaround time (TAT) = Overall time a process spent in the system

= Time of Process completion

Average Turnaround time (ATAT) = Sum of each process TAT / Total number of processes

Therefore, TAT in the system will be:

P1: 94, P2: 105, P3: 117, P4: 495, P5: 144, P6: 387, P7: 409, P8: 373, P9: 96, P10: 472.

ATAT = (94 + 105 + 117 + 495 + 144 + 387 + 409 + 373 + 96 + 472) / 10

= 2692 / 10

= 269.2

Waiting time (WT) = TAT – Burst Time

Average Waiting time (AWT) = Sum of all Process' Waiting Times / Total numbers of Processes

P1: (94 - 94) = 0, P2: (105 - 9) = 96, P3: (117 - 12) = 105, P4: (495 - 93) = 402, P5: (144 - 27) = 117, P6: (387 - 59) = 328, P7: (409 - 67) = 342, P8: (373 - 49) = 324, P9: (96 - 2) = 94, P10: (472 - 83) = 389.

AWT = (0 + 96 + 105 + 402 + 117 + 328 + 342 + 324 + 94 + 389) / 10

= 2197 / 10

= 219.7

Penalty Ratio (PR) = Turnaround time / Burst time

Average PR (APR) = Sum of PR of each process / number of processes

Variance of PR (PRV) = (Sum of squares of the PR of each process /number of processes) – APR

Penalty ratio of the respective processes will then be computed thus:

P1: (94 / 94) = 1, P2: (105 / 9) = 11.67, P3: (117 / 12) = 9.75, P4: (495 / 93) = 5.32, P5: (144 / 27) = 5.33, P6: (387 / 59) = 6.56, P7: (409 / 67) =6.10, P8: (373 / 49) = 7.61, P9: (96 / 2) = 48, P10: (472 / 83) = 5.69.

APR = (1 + 11.67 + 9.75 + 5.32 + 5.33 + 6.56 + 6.10 + 7.61 + 48 + 5.69) / 10

= 107.03 / 10

= 10.703

$PRV = ((1 + 11.67 + 9.75 + 5.32 + 5.33 + 6.56 + 6.10 + 7.61 + 48 + 5.69)^2 / 10) – 10.703^2$

= (2763.495 / 10) – 114.554209

= 276.3495 – 114.554209

= 161.795

4.5.5 Minimized Penalty Ratio Variance Algorithm (MPRV)

This is the research algorithm contributed to the field of operating systems. Its pseudocode has been clearly defined in the previous chapter. An illustration of how the algorithm works will be clearly outlined in this chapter and its results will be compared with those of already existing algorithms. Therefore, running the algorithm according to the steps in the pseudocode, still using the same values in Table 4.4 gives us the following:

Table 4.5: Processes Sorted in Increasing Order of Burst Time, with their respective EWTs and ETATs

Process	Burst time	Expected Waiting Time (EWT)	Expected Turnaround Time (ETAT)
P9	2	9	11
P2	9	38	47
P3	12	51	63
P5	27	115	142
P8	49	209	258
P6	59	252	311
P7	67	296	353
P10	83	354	437
P4	93	397	490
P1	94	401	495

TQ = median = 54

The eligibility for any process to run is based on if the expected waiting time of that process is reached or exceeded. A gantt chart based on values in Table 4.5 is generated in Fig 4.5.

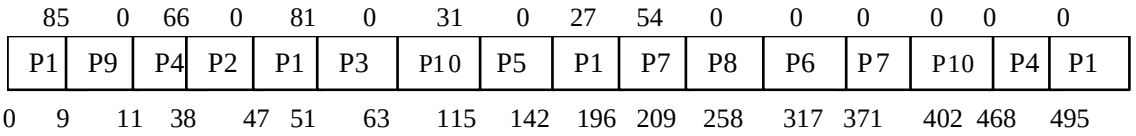


Fig 4.5: MPRV Gantt Chart

MPRV:

Waiting time (WT) = TAT – Burst Time

Average Waiting time (AWT) = Sum of all Process' Waiting Times / Total numbers of Processes

P1: (469 - 94) = 375, P2: (34 - 9) = 25, P3: (46 - 12) = 34, P4: (487 - 93) = 394, P5: (250 - 27) = 223, P6: (408 - 59) = 349, P7: (425 - 67) = 358, P8: (324 - 49) = 275, P9: (173 - 2) = 171, P10: (495 - 83) = 412.

AWT = (375 + 25 + 34 + 394 + 223 + 349 + 358 + 275 + 171 + 412) / 10

= 2616 / 10

= 261.6

Turnaround time (TAT) = Overall time a process spent in the system

= Time of Process completion – Arrival Time

Average Turnaround time (ATAT) = Sum of each process TAT / Total number of processes

Therefore, TAT in the system will be:

P1: (495 - 0) = 495, P2: (46 - 0) = 46, P3: (63 - 0) = 63, P4: (468 - 0) = 468, P5: (142 - 0) = 142, P6: (317 - 0) = 317, P7: (371 - 0) = 371, P8: (258 - 0) = 258, P9: (11 - 0) = 11, P10: (402 - 0) = 402.

ATAT = (495 + 46 + 63 + 468 + 142 + 317 + 371 + 258 + 11 + 402) / 10

= 2573 / 10

= 257.3

Penalty Ratio (PR) = Turnaround time / Burst time

Average PR (APR) = Sum of PR of each process / number of processes

Variance of PR (PRV) = (Sum of squares of the PR of each process /number of processes) – APR

Penalty ratio of the respective processes will then be computed thus:

P1: (495 / 94) = 5.27, P2: (47 / 9) = 5.22, P3: (63 / 12) = 5.25, P4: (468 / 93) = 5.03, P5: (142 / 27) = 5.26, P6: (317 / 59) = 5.37, P7: (371 / 67) = 5.54, P8: (258 / 49) = 5.27, P9: (11 / 2) = 5.5, P10: (402 / 83) = 4.84.

APR = (5.27 + 5.22 + 5.25 + 5.03 + 5.26 + 5.37 + 5.54 + 5.27 + 5.5 + 4.84) / 10

= 52.6 / 10

= 5.26

$PRV = ((5.27 + 5.22 + 5.25 + 5.03 + 5.26 + 5.37 + 5.54 + 5.27 + 5.5 + 4.84)^2 / 10) – 5.26^2$

= (276.5293/ 10) – 27.6676

= 27.70551 – 27.6676

= 0.03791

4.6 RESULTS DISCUSSION AND CONCLUSION

From our comparison with these round robin category of algorithms (RR, IRR, DQRR and NIRR), it has shown that the variance of the penalty ratio has been effectively minimized in all cases. As far as our simulations are concerned, it does not have better context switches when compared especially when the processes are fewer. It is important to note that the context switching of the RR algorithm depends on the value of the time quantum.

The IRR and NIRR algorithms generally proved to be better in Average Waiting Time, Average Turnaround Time, and Average Penalty Ratio when compared with the MPRV. This is because processes are run in ascending order of burst times in these algorithms.

The simulations were run for 10 to 5000 processes and the burst time range was 1-100.

Tables 4.6 to 4.10 show the average waiting times, average turnaround times, average penalty ratio, penalty ratio variance, and context switching values for the respective algorithms. Figs 4.6 to 4.10 show the graphical representations of the values obtained in the tables.

Table 4.6: Average Waiting Time Results Obtained

Number of Processes	RR	IRR	DQRRR	NIRR	MPRV
10	301.72	174.76	275.27	171.19	248.13
20	632.63	360.25	595.88	354.79	510.86
50	1629.08	934.88	1557.97	902.78	1299.48
100	3089.55	2011.78	3283.92	1868.92	2731.88
200	5032.03	3368.20	5027.55	6696.90	5246.63
500	16846.69	9644.46	16348.73	9288.43	13688.99
1000	32724.59	18656.86	31797.69	18046.95	26518.87
2000	67221.58	38297.97	65034.68	36926.3	55389.96
5000	166098.6	96232.43	161247.3	92095.34	136339.07

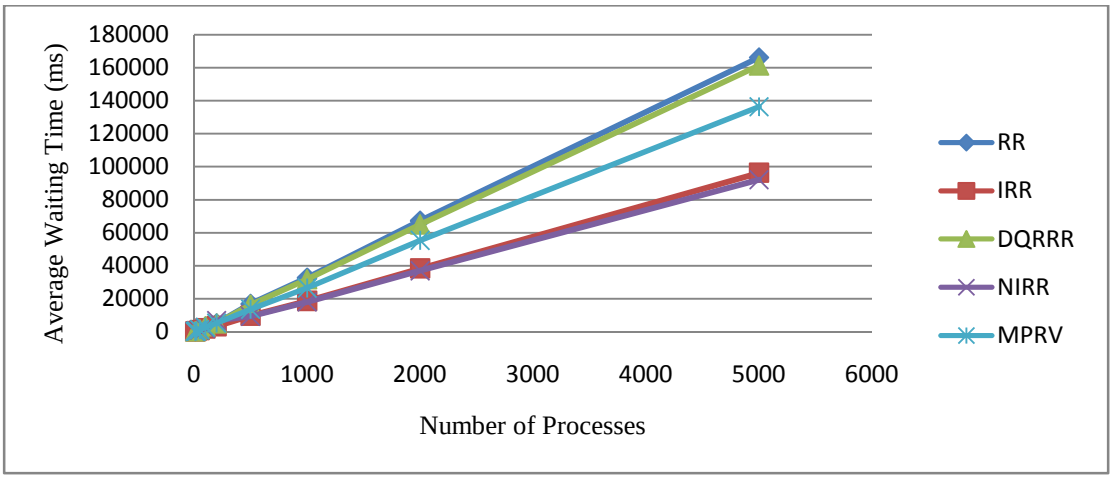


Fig 4.6: Graphical Representation of the Average Waiting Time

Table 4.6 shows the overall result of the Average Waiting Time values, while figure 4.6 represents those values graphically. From there, it is observed that the IRR and NIRR algorithms give best values. The simulations were run 100 times before the results were obtained. For each round of simulation, the same set of data was used by all of the scheduling algorithms. The range of the burst times is 1 to 100.

Table 4.7: Average Turnaround Time Results Obtained

Number of Processes	RR (TQ = 25)	IRR	DQRRR	NIRR	MPRV
10	352.62	225.67	326.17	222.09	299.04
20	682.93	410.55	646.18	405.09	561.16
50	1679.39	985.19	1608.27	953.08	1349.78
100	3269.65	1876.62	3165.13	1813.31	2574.34
200	3861.21	3861.21	6459.33	3717.14	5251.83
500	16897.52	9695.28	16399.55	9339.26	13739.82
1000	32774.36	18706.63	31847.46	18096.72	26568.64
2000	67272.11	38348.5	65085.2	36976.83	55440.49
5000	166148.9	96282.79	161297.6	92145.69	136389.42

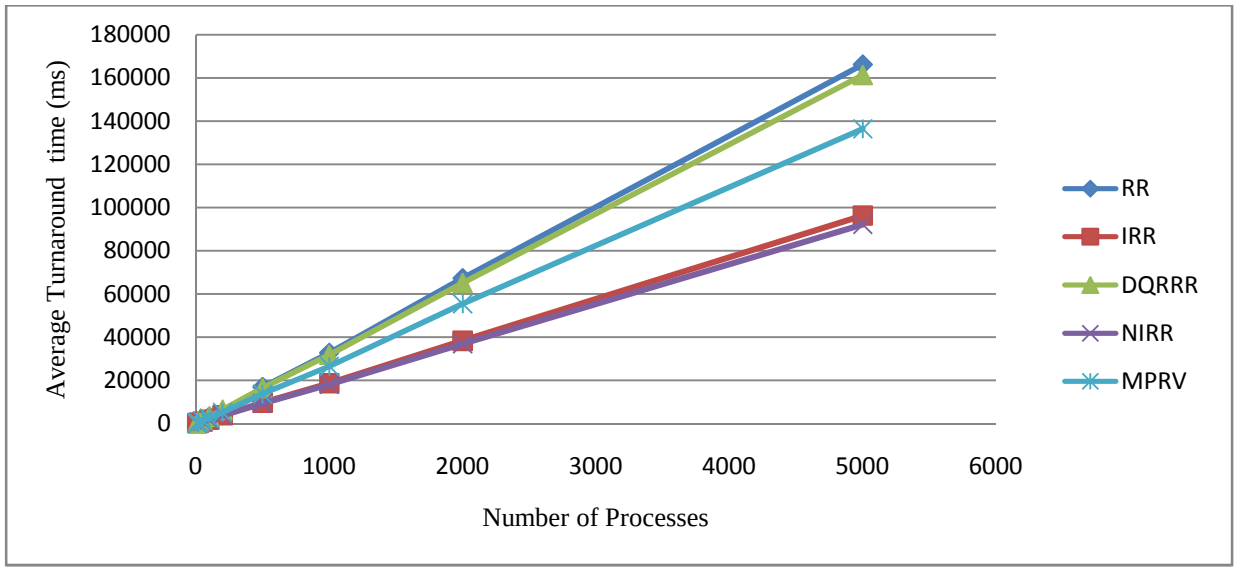


Figure 4.7: Graphical Representation of the Average Turnaround Time

Table 4.7 shows the overall result of the Average Turnaround Time values while Fig 4.7 represents those values graphically. From there it is observed that the IRR and NIRR algorithms gave best values and RR gave the worst values. The simulations were run 100 times for the results above to be obtained.

Table 4.8 shows the overall result of the Average Penalty Ratio values, while Fig 4.8 represents those values graphically. IRR generally performs best.

Table 4.8: Average Penalty Ratio Results Obtained

Number of Processes	RR (TQ = 25)	IRR	DQRRR	NIRR	MPRV
10	135.89723	3.20964	5.73684	41.35277	0.26008
20	575.91842	12.67234	18.2213	39.57906	0.5369
50	4222.68003	82.15847	67.41188	68.55054	1.64951
100	23787.3682	325.7636	213.5834	237.77138	3.65639
200	82530.0877	1322.977	680.2238	991.58121	9.79968
500	474427.905	8420.836	3396.449	6876.4558	147.4377
1000	2171564.01	31713.36	13437.33	26361.201	748.96844
2000	8123076.97	134063.0	46723.15	113990.51	4361.2102
5000	64887866.9	841599.4	324987.8	700368.77	22781.663

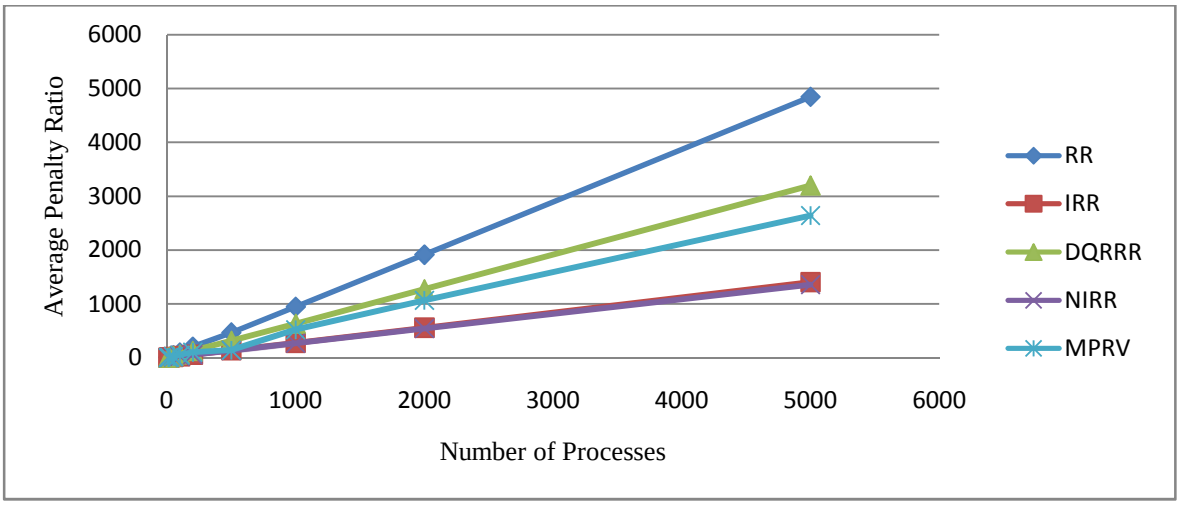


Fig 4.8: Graphical Representation of the Average Penalty Ratio

Table 4.9: Penalty Ratio Variance Results Obtained

Number of Processes	RR (TQ = 25)	IRR	DQRRR	NIRR	MPRV
10	135.89723	3.20964	5.73684	41.35277	0.26008
20	575.91842	12.67234	18.2213	39.57906	0.5369
50	4222.68003	82.15847	67.41188	68.55054	1.64951
100	23787.3682	325.7636	213.5834	237.77138	3.65639
200	82530.0877	1322.977	680.2238	991.58121	9.79968
500	474427.905	8420.836	3396.449	6876.4558	147.4377
1000	2171564.01	31713.36	13437.33	26361.201	748.96844
2000	8123076.97	134063.0	46723.15	113990.51	4361.2102
5000	64887866.9	841599.4	324987.8	700368.77	22781.663

Table 4.9 gives us values of the penalty ratio variance that were obtained from the simulations run for the different CPU scheduling algorithms.

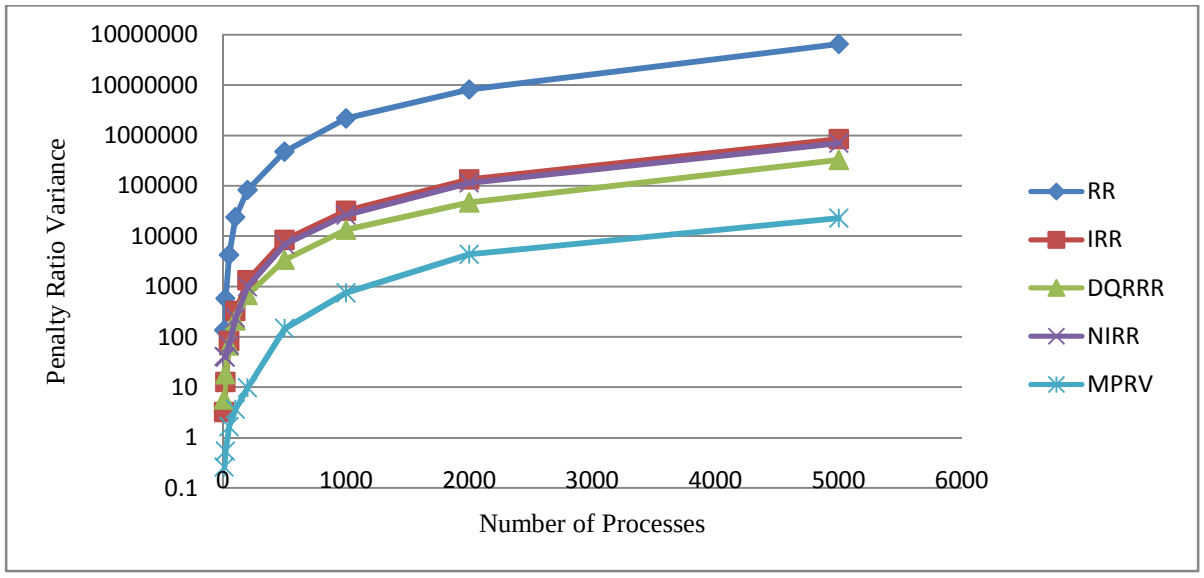


Fig 4.9: Graphical Representation of the Penalty Ratio Variance

Fig 4.9 shows that the proposed algorithm performs best in minimizing the variance of the penalty ratio. The y-axis of the graph was scaled logarithmically.

Table 4.10: Context Switching Results Obtained

Number of Processes	RR (TQ = 25)	IRR	DQRRR	NIRR	MPRV
10	24	13	18	11	14
20	50	25	37	24	28
50	125	65	95	61	70
100	251	130	192	124	136
200	506	262	386	249	266
500	1276	654	962	622	589
1000	2508	1299	1937	1245	1098
2000	5092	2608	3823	2491	2110
5000	12660	6636	9577	6259	5137

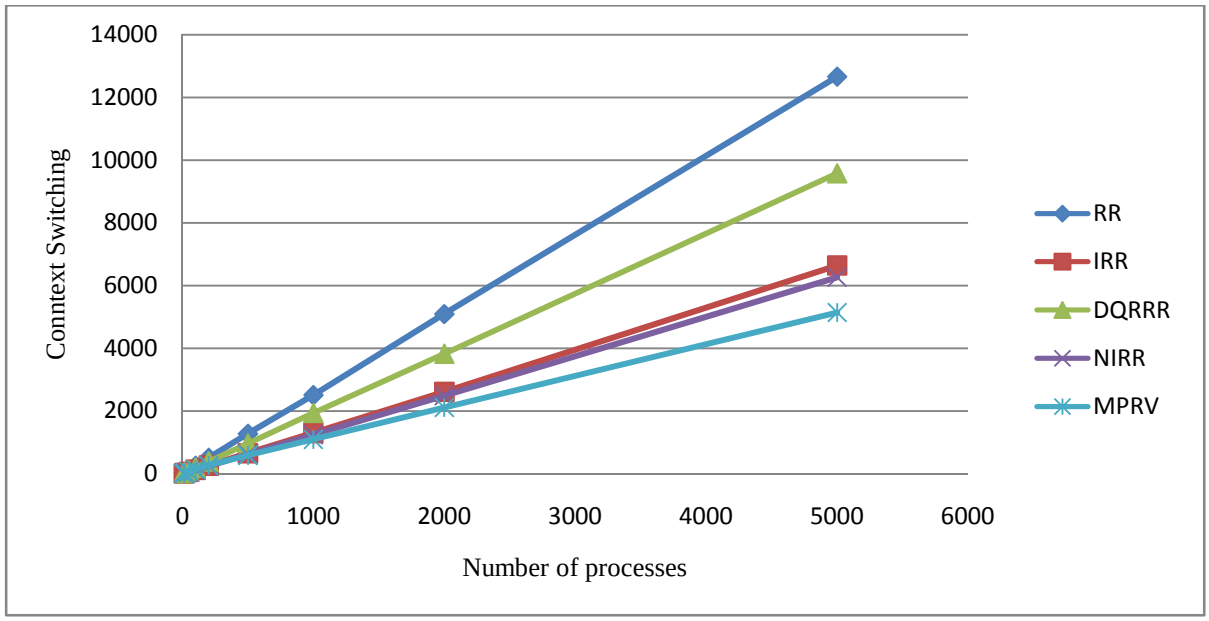


Fig 4.10: Graphical Representation of the Context Switching Values

Table 4.10 gives us a tabular representation of the context switching values for the processes. From Fig 4.10, it is inferred that the algorithm loses on context switching especially when compared with IRR and NIRR when there are fewer processes.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 INTRODUCTION

Process scheduling is a good indicator of how well an operating system performs. The choice of scheduling algorithm is therefore necessary especially as it concerns what the algorithm seeks to achieve in the course of process scheduling. This is what designers take into consideration while developing and implementing a process scheduling algorithm.

5.2 SUMMARY

The objective of the thesis was to design a scheduler that takes into cognizance the penalty ratio of the processes and seeks to minimize its variance. Since the penalty ratio is a factor of the turnaround time, by so doing, a regulated and consistent pattern in the turnaround time values is achieved. The algorithm was developed and PHP programming was used to implement and simulate the algorithm by testing with different occurrences. In the program interface, the user specifies the number of processes, the range of the burst time, the value of the time quantum (for RR scheduling algorithm) and also selects the algorithm(s) he or she desires to run or to compare especially as it concerns the variance of the penalty ratio. There are also three output tables available: the tables showing detailed results for the respective scheduling algorithms of the user's choice, the averages showing a table that displays the average of a number of runs and the summary table, which compresses the detailed tables and shows the highlights of the simulation results at a glance.

The MPRV algorithm proposed in this research encourages longer jobs to make progress (as seen in the pseudocode presented in chapter three and the Gantt charts drawn in chapter four) thereby

minimizing their starvation. It also generally has a drawback of worse waiting and turnaround time values especially when short processes are run first or there are fewer processes or the burst time range is small. The algorithm also has context switching problems especially when there are few processes in the system. However, it was discovered after repeated simulations that the proposed MPRV algorithm can compete reasonably with other classical scheduling algorithms especially when there are very many processes in the system and the burst time range is very large.

5.3 CONCLUSION

The research has answered the questions it set out to in chapter one. A framework that minimizes the variance of the penalty ratio has been developed and implemented. The algorithm was designed with an overall aim to ensure service timeliness.

5.4 FUTURE WORK

The following areas can be considered for future work as far as this research is concerned:

a. A test of the MPRV algorithm using various arrival times.
b. An alternative way of computing the value of target penalty ratio different from the one used in the research.
c. Other conditions for preemption in the algorithm.

REFERENCES

Abdullahi, I. (2012, October). Process Scheduling in Longest Job First Algorithm: A Framework for Minimizing Starvation. Ahmadu Bello University, Zaria, pp 18- 35

Abdulrahim, A., Abdullahi, S. E. and Sahalu, B. J. (2014). A New Improved Round Robin (NIRR) CPU Scheduling Algorithm. *Internation Journal of Computer Applications, 90*(4), 27 - 33.

Babu, S.B., Priyanka, N.N. and Kumar ,S.B. (2012). Efficient Round Robin CPU Scheduling Algorithm. *International Journal of Engineering Research and Development, 4*(9), 36 - 42.

Bandarupalli, S.B., Nutulapati, N. P. and Varma, P. S. (2012). A Novel CPU Scheduling Algorithm-Preemptive and Non- Preemptive. *International Journal of Modern Engineering Research, 2*(6), 4484 - 4490.

Banerjee, P., Banerjee, P., Dhal, S.S. (2012). Performance Evaluation of a New Proposed Average Mid Max Robin (AMMRR) Scheduling Algorithm with Round Robin Schedling Algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering, 2*(8), 143 - 151.

Barade, S. and Khande, N. (2010). Novel Scheduling Algorithm for Uniprocessor Operating System. *Journal for Global Research in Computer Science, 1*(2), 28 - 30.

Behera, H.S. and Swain, B.K. (2012). A New Proposed Precedence based Round Robin with Dynamic Time Quantum (PRRDTQ) Schedulinng Algorithm for Soft Real Time Systems. *International Journal of Advanced Research in Computer Science and Software Engineering, 2*(6), 223 - 227.

Behera, H.S., Swain, B.H., Parida, A.K. and Sahu, G. (2012). A new Proposed Round Robin with Highest Response Ratio Next (RRHRRN) Scheduling Algorithm for Soft Real Time Systems. *International Journal of Engineering and Advanced Technology, 1*(3), 200 - 206.

Bolesaw, S. (1990). Mutual Exclusion Revisited. *Fifth Jerusalem Conference in Information Technology*, 110-117.

Chaudhary, A., Sarwar S. and Yadav, R.S. (2012). Optimization on the Performance Of CPU Scheduling. *VSRD International Journal of Computer Science annd Infromation Technology, 2*(10), 854-856.

Curtis, S., Behera, H.S. and Panda, P. (2012). Design and Performance Evaluation of the Advance Mix Job with Dynamic Quantum Round Robin Scheduling Algorithm for Real Time Systems. *Journal for Global Research in Computer Science, 3*(3), 46 - 53.

Das, S., Goel, P., Kaur, K. (2012). A Fuzzy Approach Scheduling on More Than One

Processor System in Real Time Environment. *International Journal of Scientific Research Engineering & Technology , 1* (5), 289-293.

Dawood, A. (2012). Improving Efficiency of Round Robin Scheduling Using Ascending Quantum And Minimum-Maximum Burst Time. *Journal of University of Anbar for Pure Science,* 6 (2).

Ernst, B.W, Schroeder, B. and Urvoykella, G. (2007). Scheduling in Practice. *Journal of Parallel Processing*, *34*(4), 2-7.

Haque, A. (2012). An Analysis and Comparison of Processor Scheduling Techniques. Retrieved from http://www.cs.utexas.edu/~ahaque/downloads/processor_scheduling.pdf

Harris, J. A. (2002). *Schaum's Outlines Series: Operating Systems.* (B. Gilson.and M. Walker, Ed.) McGraw Hill, USA, pp 15 - 26.

Jaiswal, U. C. (2012). An Efficient Process Scheduling Algorithm: Long Distance Algorithm. *International Journal of Computer Science and Information Technology and Security, 2*(4), 765-767.

Jiang, S. and Vaidya, N.H. (1998). *Scheduling Algorithms for a Data Broadcast System: Minimizing the Variance of the Response time.* Technical Report, Texas A & M University, pp 5 - 18

Lenka, R.J. and Ranjan, P. (2012). A 2LFQ Scheduling with Dynamic Time Quantum using Mean Average. *International Journal of Computer Applications, 47*(23), 16 - 19.

Meehean, J. (2011). Towards Transparent CPU Scheduling. University of Wisconsin - Madison. Retrieved from research.cs.wisc.edu/ads/publications/meeheanthesis11.pdf, pp 11 - 31

Mohanty, R., Behera, H.S. 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, 5*(5), 10-15.

Mohanty, R., Behera, H.S., Patwari, K., Dash, M.,and Prasanna, M.L. (2011). Priority Based Dynamic Round Robin (PBDRR) Algorithm with Intelligent Time Slice for Real Time Systems. *International Journal of Advanced Computer Science and Applications, 2*(2), 46 - 50.

Mostafa, S.M., Rida, S.Z. and Hamad, S.H. (2010). Finding Time Quantum of Round Robin CPU SChediling Algorithm in General Computing Systems Under Integer Programming. *IJRRAS, 5*(1), 64 - 71.

Nayak, D., Malla, S.K., Debadarsh, D. (2012). An Improved Round Robin Scheduling using Dynamic Time Quantum. *International Journal of Computer Applications, 38*(5), 34-38.

Neetu, G. and Garg, R.B. (2012). Comparative Study of CPU Scheduling Algorithms. *Journal of Graphics and Image Processing, 2*(4), 245- 251.

Niu, J. (2003). Uniprocessor Scheduling. Retrieved October 26, 2013 from http://www.sci.brooklyn.cuny.edu/~jniu/teaching/csc33200/csc33200.html: http://www.sci.brooklyn.cuny.edu/~jniu/teaching/csc33200/files/1201-UniprocessorScheduling.pdf

Oyetunji, E.O. and Oluleye, A.E. (2009). Performance Assessment of some CPU Scheduling Algorithms. *Research Journal of Information Technology, 1*(1), 22-26.

Pattanayak, R., Mallick, P. and Behera, H.S. (2012). An Improved Fuzzy - Based CPU Scheduling (IFCS) Algorithm for Real Time Systems. *International Journal of Soft Computing and Engineering, 2*(1), 326 - 331.

Saxena, F.A.H. and Agarwal, S.B.P. (2012). Design and Performance Evaluation of OptimumService Time Concept for Round Robin Algorithm (OSTRR). *International Journal of Machine Learning and Computing, 2*(2), 113 - 117.

Schneiderman, B. (1984). Response Time and Display Rate in Human Performance with Computers. *Computing surveys, 16*(3), 265-285. Retrieved June 14, 2013, from http://www.cs.umd.edu/~ben/papers/Shneiderman1984Response.pdf

Sharma, V., Varshney, E.M., and Sharma, S. (2012). Design and Implementation of Operating System. Retrieved from http://books.google.com.ng/books?isbn=9380386419

Silberschatz, A., Galvin, P.B. and Gagne, G. (2005). *Operating System Concepts.* (7th Edition ed.). (B. Zobrist, K. Santor and M. Lesure, Eds.) NJ, USA: John Wiley and Sons Inc.

Srivastav, M.K., Pandey, S., Gahoi, I., and Namdev, N.K. (2012). Fair Priority Round with Dynamic Time Quanttum: FPRRDQ. *International Journal of Modern Engineering Research, 2*(3), 876 - 881.

Stallings, W. (2012). *Operating Systems Internals and Design Principles.* USA, USA: Prentice Hall.

Tanenbaum, A.S. and Woodhull, A.S. (2006). *Operating Systems: Design and Implementation.* (3rd Ed) New Jersey: Prentice Hall.

Tevfik, K. (2013). *CPU Scheduling -II.* Retrieved from http://www.cse.buffalo.edu/faculty/tkosar/cse421-521/slides/06-CPU_Scheduling_II_2spp.pdf

Yadav, R.K., Mishra, A.K., Prakash, N. and Sharma, H. (2012). An Improved Round Robin Scheduling Algorithm for CPU Scheduling. *International Journal on Computer Science and Engineering, 2*(4), 1064 - 1066.

Ye, N., Li, X., Farley, T.,and Xu, X. (2005). Job Scheduling methods for Minimizing Waiting Time Variance. *Computers and Operations Research, 34*(2007), 3069 - 3083. doi:10.1016/j.cor.2005

APPENDIX

PHP Source codes for the MPRV Algorithm

```
<?php
//Generic functions used
function generateRandomBurstTimesSorted($NumberOfProcesses, $Range)
{
   $temp = array();
   for ($i = 0; $i < $NumberOfProcesses; $i++) {
      $temp[$i] = mt_rand(1, $Range);
   }
   sort($temp);
   return $temp;
}

function generateRandomBurstTimes($NumberOfProcesses, $Range){
   $temp = array();
   for ($i = 0; $i < $NumberOfProcesses; $i++) {
      $temp[$i] = mt_rand(1, $Range);
   }
   return $temp;
}

function sortGeneratedRandomBurstTimes($generatedBurstTimes){
   sort($generatedBurstTimes);
   return $generatedBurstTimes;
```

```
}

function generateShuffledPriorities($NumberOfProcesses){
    $numbers = range(1, $NumberOfProcesses);
    shuffle($numbers);
    return $numbers;
}



function getMedianForSortedArray($data)
{
    $middle = round(count($data) / 2);
    $median = $data[$middle - 1]; //handles odd arrays

    if (count($data) % 2 == 0) {
        $median = ($data[$middle] + $data[$middle - 1]) / 2;
    }
    return $median;
}

function getBTlessThanTQUnprocessed($a, $numProcs, $oTQ){
    $b = array();
    for($i = 0; $i < $numProcs; $i++){
        if(($a[$i]->IsProcessed == FALSE) && ($a[$i]->TempBurstTime <= $oTQ)){
            $b[$i] = $a[$i]->TempBurstTime;
```

```
        }
    }
    return $b;
}

function getHBTIndex($a, $numProcs){
    $j = 0; //highest BT index
    $k = 0; //highest BT
    for ($l = 0; $l < $numProcs; $l++) {
        if($a[$l]->IsProcessed == FALSE){
            if($a[$l]->TempBurstTime >= $k){
                $k = $a[$l]->TempBurstTime;
                $j = $l;
            }
        }
    }

    return $j;
}

function getHBTValue($a, $numProcs){
    $j = 0; //highest BT index
    $k = 0; //highest BT
    for ($l = 0; $l < $numProcs; $l++) {
        if($a[$l]->IsProcessed == FALSE){
```

```
            if($a[$l]->TempBurstTime >= $k){
                $k = $a[$l]->TempBurstTime;
                $j = $l;
            }
        }
    }

    return $k;
}

function getLBTIndex($a, $numProcs){
    $j = null;
    $k = null; //lowest BT

    //init lowest BT index and value with first element in array that's unprocessed
    for ($m = 0; $m < $numProcs; $m++) {
        if($a[$m]->IsProcessed == FALSE){
            $j = $m;
            $k = $a[$m]->TempBurstTime;
        }
    }

    for ($l = 0; $l < $numProcs; $l++) {
        if($a[$l]->IsProcessed == FALSE){
            if($a[$l]->TempBurstTime < $k){
```

```
            $k = $a[$l]->TempBurstTime;
            $j = $l;
        }
    }
}

return $j;
}

function getLBTValue($a, $numProcs){
$j = null;
$k = null; //lowest BT

//init lowest BT index and value with first element in array that's unprocessed
for ($m = 0; $m < $numProcs; $m++) {
    if($a[$m]->IsProcessed == FALSE){
        $j = $m;
        $k = $a[$m]->TempBurstTime;
    }
}

for ($l = 0; $l < $numProcs; $l++) {
    if($a[$l]->IsProcessed == FALSE){
        if($a[$l]->TempBurstTime < $k){
            $k = $a[$l]->TempBurstTime;
```

```
            $j = $l;
        }
    }
}

return $k;
}

function getHPIndex($a, $numProcs){
    $j = 0; //highest Priority index
    $k = 0; //highest Priority value
    for ($l = 0; $l < $numProcs; $l++) {
        if($a[$l]->IsProcessed == false){
            if($a[$l]->AssignedPriority > $k){
                $k = $a[$l]->AssignedPriority;
                $j = $l;
            }
        }
    }

    return $j;
}

function processCondition1($OnyinyesDataArray, $key, &$runtime, $HBTIndex, $z)
{
```

```
//set processed to true
    $OnyinyesDataArray[$key]->IsProcessed = TRUE;

    //Run time += EWT - Run Time
    $runtime += ($OnyinyesDataArray[$key]->TempEWT - $runtime);

    //Set Waiting Time = Run Time
    $OnyinyesDataArray[$key]->WaitingTime = $runtime;

    //Set TAT to Run Time + BT
    $OnyinyesDataArray[$key]->TurnAroundTime = ($runtime + $OnyinyesDataArray[$key]->TempBurstTime);

    //HBT -= EWT - Run Time
    if (($OnyinyesDataArray[$key]->TempEWT - $runtime) <= 0) {
        $OnyinyesDataArray[$HBTIndex]->TempBurstTime -= $OnyinyesDataArray[$key]->TempEWT;
    } else {
        $OnyinyesDataArray[$HBTIndex]->TempBurstTime -= ($OnyinyesDataArray[$key]->TempEWT - $runtime);
    }

    //Run Time += BT
    $runtime += $OnyinyesDataArray[$key]->TempBurstTime;

    echo $z . ' done<br/>';
```

```
}


?>
```

Common fields:

```
<?php
//COMMON BITS
// 1. Process
// 2. Burst Times

//Declare Arrays to hold data for various Algorithms
$FirstComeFirstServedDataArray = array();
$RoundRobinDataArray = array();
$OnyinyesDataArray = array();
$SJFDataArray = array();
$LJFDataArray = array();
$PriorityDataArray = array();

//Important variables
$cumulativeWaitingTime = 0;
$cummulativeTurnAroundTime = 0;

//Generate random Burst Times
$randomBurstTimes = generateRandomBurstTimes($NumberOfProcesses, $Range);
```

```
//Sorted burst times
$randomSortedBurstTimes = sortGeneratedRandomBurstTimes($randomBurstTimes);

$shuffledPriorities = generateShuffledPriorities($NumberOfProcesses);

for ($i = 0; $i < $NumberOfProcesses; $i++) {

    //For Round Robin Algorithm
    $t = new tableRow();

    $t->Id = $i + 1;
    $t->ProcessName = "P" . ($i + 1);
    $t->BurstTime = $randomBurstTimes[$i];
    $t->IsProcessed = false;
    $t->TempBurstTime = $t->BurstTime;
    $t->WaitingTime = 0;
    $t->TurnAroundTime = 0;
    $t->PenaltyRatio = "";
    $t->HasOverFlow = false;
    $t->NumberOfOverflows = 0;
    $t->OverflowRemainder = 0;

    $RoundRobinDataArray[$i] = $t;
```

```
//For MPRV Algorithm
$u = new tableRow();

$u->ProcessName = "P" . ($i + 1);
$u->BurstTime = $randomSortedBurstTimes[$i];
$u->TempBurstTime = $u->BurstTime;
$u->IsProcessed = false;
$u->DummyBurstTime = $randomBurstTimes[$i];

$OnyinyesDataArray[$i] = $u;



//For SJF
$s = new tableRow();

$s->ProcessName = "P" . ($i + 1);
$s->BurstTime = $randomBurstTimes[$i];
$s->IsProcessed = FALSE;
$s->TempBurstTime = $s->BurstTime;
$s->WaitingTime = 0;
$s->TurnAroundTime = 0;
$s->PenaltyRatio = "";

$SJFDataArray[$i] = $s;
```

```
//For LJF
$l = new tableRow();

$l->ProcessName = "P" . ($i + 1);
$l->BurstTime = $randomBurstTimes[$i];
$l->IsProcessed = FALSE;
$l->TempBurstTime = $l->BurstTime;
$l->WaitingTime = 0;
$l->TurnAroundTime = 0;
$l->PenaltyRatio = "";

$LJFDataArray[$i] = $l;


//For Priority
$p = new tableRow();

$p->ProcessName = "P" . ($i + 1);
$p->BurstTime = $randomBurstTimes[$i];
$p->IsProcessed = FALSE;
$p->TempBurstTime = $p->BurstTime;
$p->WaitingTime = 0;
$p->TurnAroundTime = 0;
$p->PenaltyRatio = "";
```

```
    $p->AssignedPriority = $shuffledPriorities[$i];

    $PriorityDataArray[$i] = $p;

        //For FCFS Algorithm
    $f = new tableRow();

    $f->Id = $i + 1;
    $f->ProcessName = "P" . ($i + 1);
    $f->BurstTime = $randomBurstTimes[$i];
    $f->IsProcessed = FALSE;
    $f->WaitingTime = 0;
    $f->TurnAroundTime = 0;
    $f->PenaltyRatio = "";

    $FirstComeFirstServedDataArray[$i] = $f;

}
?>
```

MPRV algorithm:

```
<?php

//MPRV Algorithm
```

```
//Calculate Target Penalty Ratio for Onyinye's Algorithm
$TargetPR = 0;
$SumOfBurstTimes = 0;
$tempArray = array();

$RuntimeArray = array();

for ($i = 0; $i < $NumberOfProcesses; $i++) {
   $SumOfBurstTimes = $OnyinyesDataArray[$i]->BurstTime + $SumOfBurstTimes;
   $tempArray[$i] = $OnyinyesDataArray[$i]->BurstTime;
}

$HighestBT = max($tempArray);
$TargetPR = round($SumOfBurstTimes / $HighestBT, 2);

//Populate Data Fields
for ($i = 0; $i < $NumberOfProcesses; $i++) {
   $OnyinyesDataArray[$i]->ETAT = round($OnyinyesDataArray[$i]->BurstTime *
$TargetPR); //ETAT
   $OnyinyesDataArray[$i]->EWT = $OnyinyesDataArray[$i]->ETAT -
$OnyinyesDataArray[$i]->BurstTime; //EWT
   $OnyinyesDataArray[$i]->TempWaitingTime = $OnyinyesDataArray[$i]->EWT;
   $OnyinyesDataArray[$i]->TempEWT = $OnyinyesDataArray[$i]->EWT;
   $OnyinyesDataArray[$i]->TempBurstTime = $OnyinyesDataArray[$i]->BurstTime;
}
```

```
$md = getMedianForSortedArray($randomSortedBurstTimes);

//Time Quantum
$oTQ = round($md);

//START

//1. Set Run Time and TAT to 0

$runtime = 0;
$tat = 0;

for($a = 0; $a < $NumberOfProcesses; $a++){
    //cache contents of array at that index
    $b = $OnyinyesDataArray[$a];

    //var_dump($b);
test1loopback:
    //TEST 1: Is TempEWT - RT lte 0?
    if(($b->TempEWT - $runtime) <= 0){
        //YES
test4loopback:

        $d = count($RuntimeArray);
        $RuntimeArray[$d] = new RTCapture();
```

```
    $c = $RuntimeArray[$d];
    $c->ProcessName = $b->ProcessName;
    $c->PreRT = $runtime;

    $b->IsProcessed = True;
    $b->WaitingTime = $runtime; //there's a question mark here
    $runtime += $b->TempBurstTime;
    $tat = $runtime;
    $b->TurnAroundTime = $tat;

    $c->PostRT = $runtime;

} else{
    //NO

    //TEST 2: Is EWT - RT lte TQ?
    if(($b->TempEWT - $runtime) <= $oTQ){

        //YES

        //Get HBT Process Index
        $HBTIndex = getHBTIndex($OnyinyesDataArray, $NumberOfProcesses);
        if($OnyinyesDataArray[$HBTIndex]->TempBurstTime == $b->TempBurstTime){
            goto test4loopback;
        } else{
```

```
//TODO: Add HBT Process data as well
$e = count($RuntimeArray);
$RuntimeArray[$e] = new RTCapture();
$f = $RuntimeArray[$e];

//Add current process data to RT array
$d = count($RuntimeArray);
$RuntimeArray[$d] = new RTCapture();
$c = $RuntimeArray[$d];
$c->ProcessName = $b->ProcessName;

$b->IsProcessed = True;
//Get HBT Process Index
$HBTIndex = getHBTIndex($OnyinyesDataArray, $NumberOfProcesses);
//Modify
$OnyinyesDataArray[$HBTIndex]->TempBurstTime -= ($b->TempEWT - $runtime);
$f->ProcessName = $OnyinyesDataArray[$HBTIndex]->ProcessName;
$f->PreRT = $runtime;
$f->PostRT = $f->PreRT + ($b->TempEWT - $runtime);

//Update Run Time
$runtime += ($b->TempEWT - $runtime);

$c->PreRT = $runtime;
```

```
            //Update Waiting Time
            $b->WaitingTime = $runtime;
            //Update TAT
            $tat = $runtime + $b->TempBurstTime;
            $b->TurnAroundTime = $tat;
            //Update Run Time
            $runtime += $b->TempBurstTime;

            $c->PostRT = $runtime;
        }

    } else{
        //NO
test3loopback:
        //TEST 3: Is (Temp)EWT - TAT lte TQ?
        if(($b->TempEWT - $tat) <= $oTQ){
            //YES

            //Get HBT Process Index
            $HBTIndex = getHBTIndex($OnyinyesDataArray, $NumberOfProcesses);
            if($OnyinyesDataArray[$HBTIndex]->TempBurstTime == $b->TempBurstTime){
                goto test4loopback;
            } else{
```

```
        $d = count($RuntimeArray);
        $RuntimeArray[$d] = new RTCapture();
        $c = $RuntimeArray[$d];
        //$c->ProcessName = $b->ProcessName;
        $c->PreRT = $runtime;

        //Get HBT
        $HBTIndex = getHBTIndex($OnyinyesDataArray, $NumberOfProcesses);
        //Modify
        $OnyinyesDataArray[$HBTIndex]->TempBurstTime -= $b->TempEWT;

        $c->ProcessName = $OnyinyesDataArray[$HBTIndex]->ProcessName;

        $runtime += $b->TempEWT;
        $b->TempEWT = $runtime;

        $c->PostRT = $runtime;

        goto test1loopback;
    }

}else{
    //NO

    //Get HBT Process Index
```

```
$HBTIndex = getHBTIndex($OnyinyesDataArray, $NumberOfProcesses);
if($OnyinyesDataArray[$HBTIndex]->TempBurstTime == $b->TempBurstTime){
   goto test4loopback;
} else{

   $d = count($RuntimeArray);
   $RuntimeArray[$d] = new RTCapture();
   $c = $RuntimeArray[$d];
   //$c->ProcessName = $b->ProcessName;
   $c->PreRT = $runtime;

   //Get HBT
   $HBTIndex = getHBTIndex($OnyinyesDataArray, $NumberOfProcesses);
   //Modify
   $OnyinyesDataArray[$HBTIndex]->TempBurstTime -= $oTQ;
   $c->ProcessName = $OnyinyesDataArray[$HBTIndex]->ProcessName;

   $runtime += $oTQ;
   $b->TempEWT = $b->EWT - $runtime;
   $b->WaitingTime = $runtime;

   $c->PostRT = $runtime;

   goto test3loopback;
}
```

```
            }
        }
    }
}

for ($i = 0; $i < $NumberOfProcesses; $i++) {
    //cache contents of array at that index
    $p = $OnyinyesDataArray[$i];

    //get the processname from cached object at that index
    $processName = $p->ProcessName;



    //get all occurences of processname in RuntimeArray and store in cacheArray
    $cacheArray = array();
    foreach ($RuntimeArray as $location){
        if($location->ProcessName == $processName){
            $size = count($cacheArray);
            $cacheArray[$size] = $location;
        }
    }

    if(count($cacheArray) > 1){
        //reset WT and TAT for Process to 0, to remove any previous calculations
        $p->WaitingTime = 0;
```

```
        //$p->TurnAroundTime = 0;   //commented out because she said TAT is correct

        //add the first PreRT to current Process' WT
        $p->WaitingTime += $cacheArray[0]->PreRT;

        //loop through contents of cacheArray and perform necessary calculations
        for($k = 0; $k < count($cacheArray); $k++){

            //check if next array key exists
            $nextExists = array_key_exists($k + 1, $cacheArray);

            //if it exists, do the math
            if($nextExists){
                $p->WaitingTime += $cacheArray[$k + 1]->PreRT - $cacheArray[$k]->PostRT;
            }
        }
    }else{
        //only one element in cacheArray so assign Pre and Post to WT and TAT
        $p->WaitingTime = $cacheArray[0]->PreRT;
        $p->TurnAroundTime = $cacheArray[0]->PostRT;
    }

    //Assign Penalty Ratio
    $OnyinyesDataArray[$i]->PenaltyRatio = $OnyinyesDataArray[$i]->TurnAroundTime /
$OnyinyesDataArray[$i]->BurstTime;
```

```
    $OnyinyesDataArray[$i]->PenaltyRatio = round($OnyinyesDataArray[$i]->PenaltyRatio, 2);


    //var_dump($cacheArray);
}


//var_dump($RuntimeArray);
$contextSwitch = count($RuntimeArray) - 1;
?>
<div class="result-section">
    <h4>Simulation Results for Minimized Penalty Ratio Variance Algorithm</h4>
    <table class="table table-condensed table-bordered table-striped">
        <thead>
        <tr>
            <th>Process</th>
            <th>Burst Times</th>
            <th>Sorted Burst Times</th>
            <th>Waiting Times</th>
            <th>Turn Around Times</th>
            <th>Penalty Ratio</th>
        </tr>
        </thead>
        <tbody>
        <?php
                    echo 'Context Switch = ' . $contextSwitch;
```

```
foreach ($OnyinyesDataArray as $row) {
    echo '<tr>';
    echo "<td class='text-left'>{$row->ProcessName}</td>";
    echo "<td>{$row->DummyBurstTime}</td>";
    echo "<td>{$row->BurstTime}</td>";
    echo "<td>{$row->WaitingTime}</td>";
    echo "<td>{$row->TurnAroundTime}</td>";
    echo "<td>{$row->PenaltyRatio}</td>";
    echo '</tr>';
}
?>
<tr class="green">
    <td colspan="3">
        <b>AVERAGES =</b>
        <?php
        $WT2Sum = 0;
        $TAT2Sum = 0;
        $PR2Sum = 0;
        foreach ($OnyinyesDataArray as $row) {
            $WT2Sum += $row->WaitingTime;
            $TAT2Sum += $row->TurnAroundTime;
            $PR2Sum += $row->PenaltyRatio;
        }
        ?>
    </td>
```

```
        <td><?php echo round($WT2Sum / $NumberOfProcesses, 2); ?></td>
        <td><?php echo round($TAT2Sum / $NumberOfProcesses, 2); ?></td>
        <td><?php echo round($PR2Sum / $NumberOfProcesses, 2); ?></td>
      </tr/>
      <tr class="green">
        <td colspan="5">
          <b>VARIANCE =</b>
        </td>
        <?php
                                $PR2Sum = 0;
          $SummationPR2Sum = 0;
          foreach ($OnyinyesDataArray as $row) {
                                $PR2Sum += $row->PenaltyRatio;
                                $SummationPR2Sum +=(($row->PenaltyRatio) * ($row->PenaltyRatio));
          }
                        ?>
                        <td>
                        <?php echo round(($SummationPR2Sum / $NumberOfProcesses) - (($PR2Sum / $NumberOfProcesses) * ($PR2Sum / $NumberOfProcesses) ) , 5); ?>
                        </td>
      </tr>
      </tbody>
    </table>
    <?php
    echo 'Time Quantum = Median Value = ' . $oTQ;
```

```
    echo '<br />';
    echo "Target PR = {$TargetPR}";
    ?>
</div>
```