IMPROVED DYNAMIC TIME SLICE ROUND ROBIN SCHEDULING ALGORITHM WITH UNKNOWN BURST TIME

BY

Nurat YUSUF
P13SCMT8082

DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF PHYSICAL SCIENCES
AHMADU BELLO UNIVERSITY,
ZARIA, NIGERIA

APRIL, 2017

IMPROVED DYNAMIC TIME SLICE ROUND ROBIN SCHEDULING ALGORITHM WITH UNKNOWN BURST TIME

BY

Nurat YUSUF
P13SCMT8082

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA IN PARTIAL FULFILLMENT FOR THE AWARD OF A MASTER OF SCIENCE DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF PHYSICAL SCIENCES
AHMADU BELLO UNIVERSITY,
ZARIA, NIGERIA

APRIL, 2017

DECLARATION

I declare that the work in this dissertation proposal titled "IMPROVED DYNAMIC TIME SLICE ROUND ROBIN SCHEDULING ALGORITHM WITH UNKNOWN BURST TIME" has been carried out by me in the Department of Computer Science. The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation proposal was previously presented for another degree or diploma at this or any other Institution.

NURAT YUSUF	________________	________________
Name of Student	***Signature***	***Date***

CERTIFICATION

This dissertation entitled "IMPROVED DYNAMIC TIME SLICE ROUND ROBIN SCHEDULING ALGORITHM WITH UNKNOWN BURST TIME" by **Nurat Yusuf** meets the regulations governing the requirement for seminar defense.

____________________________	____________________
Dr. K. A. Bakare Chairman Supervisory Committee	Date
____________________________	____________________
Prof. A. A. Obiniyi Member Supervisory Committee	Date
____________________________	____________________
Prof. S. B Junaidu Head of Department	Date
____________________________	____________________
Name & Signature External Examiner	Date
____________________________	____________________
Name & Signature Dean, School of Postgraduate Studies	Date

DEDICATION

The dissertation is dedicated to God Almighty for his mercy upon my life throughout the course to this study.

ACKNOWLEDGEMENT

My gratitude goes to God almighty for his mercy, kindness and faithfulness in my life and for seeing me through the study, it is not by my power but to him alone I return all glory.

My profound gratitude goes to my Supervisor Dr. K. A. Bakare and Prof. A. A. Obiniyi who guided me throughout the period of writing this dissertation and without their directions, contribution and encouragement this dissertation will not have been written.

Special thanks go to my parent for their prayers and word of encouragement throughout my course of study, indeed their prayers goes a long away as regard to this study, may Almighty Allah grant them long life, prosperity and grant them AljanaFirdaus hereafter, amin.

This work will not be completed if I fail to say thank to my able Lecturer Dr. A. F. D. Kana for his advice and encouragement, you are indeed a mentor. I will also thank Mr. AbdulrasaqAbdulrahim for his support and to other fellow colleagues, Precious, Fatima and RilwanSalahudeen for their word of encouragement.

Finally, I appreciate my colleagues in my place of work who gave me support throughout the course of my study, may Almighty God reward you all.

ABSTRACT

Round-Robin (RR) is one of the algorithms employed by process and network schedulers in computing. As the term is generally used, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority. Most literature on Round Robin assumed burst time of processes using different statistical distributions. Dynamic Time Slice Round Robin with Unknown Burst Time was designed to solve the problem of burst timeand improve the performance of round robin but determination of burst time can only be achieved when processes have executed in first or second cycle. This dissertation proposed to improve the performance of the existing algorithms by introducing a technique that generates burst time using instruction count and number of operators. The proposed algorithm was implemented and benchmarked against First-Come-First-Serve (FCFS), Shortest-Job-First (SJF), Round Robin (RR), Improved Round Robin (IRR) and Dynamic Time Slice Round Robin Scheduling Algorithm with unknown Burst Time. Evaluation analysis shows that the proposed algorithm is better than the existing scheduling algorithms. The performance is improved by 20-25%. The results of the proposed algorithm outperformed other algorithms in term of Average Waiting Time, Number of Context Switch and Turnaround Time, but CPU utilization was not considered.

TABLE OF CONTENTS

TITLE PAGE……………………………………………………………………………i
DECLARATION ii
CERTIFICATION iii
DEDICATION iv
ABSTRACT vi
TABLE OF CONTENTS vii
CHAPTER ONE 1
INTRODUCTION 1
1.1 Background to the Study 1
1.2 Problem Statement 4
1.3 Research Motivation 4
1.4 Aim and Objectives 5
1.5 Research Methodology 5
1.6 Limitation of the Study 6
1.7 Organization of the Dissertation 6
CHAPTER TWO 7
LITERATURE REVIEW 7
2.1 Introduction 7
2.2 CPU Scheduling 7
2.3 Multiprogramming 9
2.4 CPU scheduling algorithms 11
2.5 Process 13
2.6 Queuing Theory 14
2.6.1 Characteristics of Queuing Model**...15**
2.6.2 Performance Measures in Queuing Model **..16**
2.7 CPU Time 17

2.7.1 Instruction Count 18
2.7.2 Cycle per Instruction 20
2.7.3 Clock Rate 20
2.8 Related Works 20
CHAPTER THREE..37
DESIGN OF IMPROVED DYNAMIC TIME SLICE ROUND ROBIN SCHEDULING ALGORITHM WITH UNKNOWN BURST TIME 37
3.1 Introduction 37
3.2 The Proposed Dynamic Time Slice Round Robin with Unknown Burst Time (DTSRRUBT) Scheduling Algorithm...37
3.2.1 The pseudo code of proposed Dynamic Time Slice Round Robin CPU Scheduling Algorithm with Unknown Burst Time 39
3.2.2 The flow chart 40
3.3 How the Proposed Algorithm Works 42
CHAPTER FOUR...44
IMPLEMENTATION, RESULTS AND DISCUSSIONS 44
4.1 Introduction 44
4.2 Implementation 44
4.3 System Requirement..51
4.3.1 Experimental Setup 51
4.4 System Architecture 52
4.5 Evaluation of Performance 54
4.6 Comparative Analysis..55
CHAPTER FIVE 52
SUMMARY, CONCLUSION AND RECOMMENDATION...59
5.1 Summary...59
5.2 Conclusion 59
5.3 Recommendation...60
5.4 Contribution to knowledge 60

Reference 72
Appendix..76

LIST OF TABLES

Table 3.1 Instruction Type 37

Table 3.2 Illustrative table 41

Table 4.1 Process Table..44

Table 4.2 First Come First Serve 45

Table 4.3 Shortest Job First 46

Table 4.4 Round Robin 47

Table 4.5 Improved Round Robin 48

Table 4.6 Dynamic Time Slice Round Robin...49

Table 4.7 Improved Dynamic Time Slice Round Robin...50

Table 4.8 Comparative Table using 5 processes...55

Table 4.9 Comparative Table using 100 processes...55

Table 4.10 Comparative Table using 1000 processes...56

LIST OF FIGURES

Figure 2.1:CPU Process State (Abdulrazaq *et al.* 2014)....14

Figure 2.2:Generic CPU Machine Instruction Processing Steps....14

Figure 2.3: Existing Algorithm (Angu, *et al.,* 2016)....35

Figure 3.1: The Flow Chart of the Proposed DTSRR....41

Figure 4.1: IDTSRR Interface....45

Figure 4.2:First Come First Serve Graph....45

Figure 4.3:Shortest Job First....46

Figure 4.4:Round Robin....47

Figure 4.5: Improved Round Robin....48

Figure 4.6:Dynamic Time Slice Round Robin....49

Figure 4.7:Improved Dynamic Time Slice Round Robin....50

Figure 4.11:Systerm Architecture....54

Figure 4.8:Graph of Evaluation using 5 Processes....55

Figure 4.9:Graph of Evaluation using 100 Processes....56

Figure 4.10:Graph of Evaluation using 1000 Processes....56

LIST OF ABBREVIATIONS

OS:	Operating System
RR:	Round Robin
CPU:	Central Processing Unit
I/O:	Input Output
FIFO:	First In First Out
FCFS:	First Come First Serve
SJF:	Shortest Job First
PS:	Priority Scheduling
IC:	Instruction Count
CPI:	Clock Per Instruction
AWT:	Average Waiting Time
ATAT:	Average Turnaround Time
ART:	Average Response time
NSC:	Number of Context Switch
QT:	Quantum Time
BT:	Burst Time
LJF:	Longest Job First
DRRCP:	Dynamic Round Robin with Controlled Preemption
HLVQTRR:	Half Life Varying Quantum Time Round Robin
DTSRRUBT:	Dynamic Time Slice Round Robin with Unknown Burst Time
IDTRRUBT:	Improved Dynamic Time Slice Round Robin with Unknown Burst Time

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

An operating system (OS) is the brain of a computer system who constantly and continuously manages the resources available around the system in optimum way. It is a software that controls the execution of many other application programs and acts as an interface between computer hardware and applications. It has some attractive features like multiprogramming, multitasking and multi-users, which place it way ahead in the race with human mind. One of the basic and most important tasks an OS needs to perform is job scheduling where many processing requests arrive from multiple channels to a ready queue and the system manages all in a way to achieve high efficiency level (Sendre, 2015).

In the beginning programmers needed a way to handle complex input/output operations. The evolution of computer programs and their complexities requires new necessities. Because machines began to become more powerful, the time a program needed to run decreased. However, the time needed for handling off the equipment between different programs became evident and this led to program like Disk Operating System (DOS). This confirms that operating systems were originally made to handle these complex input/output operations like communicating among a variety of disk drives.

Earlier computers were not as powerful as they are today. In the early computer systems you would only be able to run one program at a time. For instance, you could not be typing and browsing the internet all at the same time. However, today's operating systems

are very capable of handling not only two but multiple applications at the same time. In fact, if a computer is not able to do this, it is considered useless by most computer users. In order for a computer to be able to handle multiple applications simultaneously, there must be an effective way of using the CPU. Several processes may be running at the same time, so there has to be some kind of order to allow each process to get its share of CPU time.

An operating system must allocate computer resources among the potentially competing-requirements of multiple processes (Silberschatz*et al.,* 2013). In the case of the processor, the resource to be allocated is execution time on the processor and the means of allocation is scheduling. The scheduling function must be designed to satisfy a number of objectives, including fairness, lack of starvation of any particular process, efficient use of processor time, and low overhead. In addition, the scheduling function may need to take into account different levels of priority or real-time deadlines for the start or completion of certain process.

Operating system should allow processes as many as possible running at all times in order to maximize the CPU utilization. In a multi-programmed operating system a process is executed until it must wait for the completion of some I/O request. In this case, the time has been used proficiently. A number of processes are kept in memory simultaneously while one process occupies the CPU selected by the Operating System (Sharma *et al*., 2010).

Ishwari and Deepa (2012) define scheduling as a fundamental operating system function that determines which process run, when there are multiple runnable processes. CPU scheduling is important because of its impacts in resource utilization and other performance parameters. For this, scheduling algorithms are required to have pre-set systematic steps. In wider sense, one can assume random behavior of the scheduler as an extension to usual, in light of algorithm, and at the certainty level the specific algorithm or many similar may be obtained. There are many conventional CPU scheduling algorithms to determine the order in which processes are attended to such as First-Come-First-Serve (FCFS), Shortest Job First Scheduling, Round Robin scheduling, Priority Scheduling are examples of scheduling algorithm, but due to a number of disadvantages these are rarely used in real time operating systems except Round Robin scheduling. This has made Round Robin one of the most researched scheduling algorithms.

Round Robin is designed especially for time sharing system. It is a preemptive version of FCFS algorithm. To implement RR scheduling, ready queue is kept as FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after one (1) time quantum, and dispatches the process. Different approaches were used for round robin scheduling but their focus is based on calculating quantum time with assumed burst time.

Thisdissertation will introduce Instruction Count (IC) to determine estimated time of each process and quantum time will be calculated as new process arrives the ready queue.

1.2 Problem Statement

Over the years, scheduling has been the focus of intensive research, and many different algorithms such as First Come First Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling (PS) have been implemented. Today, the emphasis in scheduling research is on Round Robin where multiple processes exist concurrently in the main memory with each process alternating using a processor and waiting for some event to occur. However, many approaches used focused on determination of time quantum for efficient scheduling algorithm. Anju, *et al*. (2016) proposed an algorithm, Dynamic Time Slice Round Robin with Unknown Burst Time (DTSRRUBT) to determine burst time of processes by assuming a value as initial time quantum, burst time is determined from the finished processes after execution for one cycle. However, the problem here is that the processes on queue have to run for one cycle while burst time is determined from those processes that finished with the initial time quantum. Therefore, this research modifies DTSRRUBT to determine unknown burst time using instruction count to improve the performance of CPU scheduling.

1.3 Research Motivation

CPU scheduling is one of the challenging issues in operating system design. To resolve this, different scheduling algorithms have been proposed. These scheduling algorithms required some parameter such as arrival time, burst time and quantum time which enable the scheduler to predict the behavior of possible processes. Prior to the execution of a process, the burst time is not known. However, most researchers use different statistical distributions such as normal distribution, uniform distribution and exponential

distribution to assume burst times. Hence, there is need to address the issue of assumption of burst times in order to help a scheduler efficiently allocate processes to CPU.

1.4 Aim and Objectives

The aim of this dissertation is to enhance Dynamic Time Slime Round Robin with Unknown Burst Time (DTSRRUBT).

The objectives are to:

a. design an algorithm based on the modification of Dynamic Time Slice Round Robin Scheduling Algorithm with Unknown Burst Timedeveloped by Anju*et al*., 2016;

b. Implement the improved version of the algorithm in (a);

c. Analyze and evaluate the result with other algorithms (i.e. FCFS, SJF, IRR, NIRR and DTSRRBT) in term of AWT, ATAT, ART and NCS.

1.5 Research Methodology

The followings are steps adopted for this research work:

a. Review of literatures on CPU scheduling algorithms especially in the area of Round Robin scheduling algorithm.

b. Design of new algorithm to improve DTSRRBT by using:

    i. Instruction count and number of operators to generate burst time

    ii. Average burst time as quantum time.

c. Simulation of the new algorithm and other algorithms such as FCFS, SJF, RR, IRR, NIRR and DTSRRUBT using Java FX.

d. The seven algorithms (i.e. FCFS, SJF, RR, IRR, NIRR, DTSRRBT and the modified IDTSRRBT) are compared based on Average Waiting time, Average Turnaround Time, Average Response Time and Number of Context Switch.

1.6 Limitation of the Study

Improved Dynamic Time Slice Round Robin Scheduling Algorithm with Unknown burst is designed to address the issue of assumption of burst time and processes are executed using dynamic quantum time. The algorithm is tested on Average Waiting Time, Number of Context Switch, Turnaround Time and Response time, but CPU utilization was not considered.

1.7 Organization of the Dissertation

In chapter two, we present the concept of CPU scheduling algorithm, criteria of CPU scheduling, meaning of process, queuing theory, CPU time, instruction count and the review of related literature, while Chapter three focuses on the methodology adopted to the design, the proposed algorithm such as the model used to calculate burst time, pseudo code, flow chat and how the proposed algorithm work.. In Chapter Four, we discuss the result of the proposed IDTSRRcompared with other algorithm such as FCFS, SJF, RR, IRR and DTSRR. Finally, chapter five summarizes the study and presents the conclusion, recommendation and future work.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter discusses the concepts of CPU scheduling in operating systems, scheduling criteria, types of scheduling algorithm, queuing theory, meaning of process, CPU time, instruction count and clock rate. It also discusses the work done by various researchersespecially in the area of round robin algorithm

2.2 CPU Scheduling

CPU scheduling is the basis of multiprogramming operating systems, by switching the CPU among processes. The operating system can make the computer more productive, whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler, or CPU scheduler. The scheduler selects a process from the processes in memory that are ready to execute and allocate the CPU to that process (Silberschatz*et al*. 2013).

In a single-processor system, only one process can run at a time, any other must wait until the CPU is free and can be rescheduled. (Ajala*et al.,* 2015) the objective of multiprogramming, is to have some processes running at all times, to maximize CPU utilization. Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, scheduling is central to operating-system design. CPU scheduling determines which process run when there are multiple run-able processes. A good CPU

scheduling techniqueensures adequate resource allocation and management at minimal possible time.

2.2.1 Scheduling Criteria

There are various CPU scheduling algorithms which have different properties, and the choice of a particular algorithm may favour one class of processes over another. For selection of an algorithm for a particular situation, we must consider properties of various algorithms. The scheduling criteria include the following (Ishwari and Deepa, 2012).

a. **Context Switch:** A context switch is process of storing and restoring context (state) of a preempted process, so that execution can be resumed from same point at a later time. Context switching is usually computationally intensive, lead to wastage of time and memory, which in turn increases the overhead of scheduler, so the design of operating system is to optimize only these switches.

b. **Throughput:** Throughput is defined as number of processes completed per unit time. Throughput is slow in round robin scheduling implementation. Context switching and throughput are inversely proportional to each other.

c. **CPU Utilization:** This is a measure of how much busy the CPU is. Usually, the goal is to maximize the CPU utilization.

d. **Turnaround Time**: Turnaround time refers to the total time which is spent to complete the process and how long it takes the time to execute that process. The time interval from the time of submission of a process to the time of completion is the turnaround time. Total turnaround time is the sum of the periods spent waiting to get into memory, waiting time in the ready queue, execution time on the CPU and doing I/O.

e. **Waiting Time**: Waiting time is the total time a process has been waiting in ready queue. The CPU scheduling algorithm does not affect the amount of time during which a process executes or does input-output; it affects only the amount of time that a process spends waiting in ready queue.

f. **Response Time:** In an interactive system, turnaround time may not be best measure. Often, a process can produce some output fairly early and can continue computing new results while previous results are being produced to the user. Thus, response time is the time from the submission of a request until the first response is produced. That means,with time when the task is submitted until the first response is received. So the response time should be low for best scheduling.

2.3 Multiprogramming

Multiprogramming systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user's interaction with the computer system. Time sharing (or multitasking) is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.(Gupta and Rajput, 2012).

OS may feature up to three distinct types of schedulers: a long term scheduler (also known as an admission scheduler or high level scheduler), a mid-term or medium-term scheduler and a short-term scheduler (also known as a dispatcher or CPU scheduler).

a. **Long-term Scheduler**: The long-term or admission scheduler decides which job or processes are to be admitted to the ready queue, that is, when an attempt is made to execute a process, its admission to the set of currently executing process is either authorized or delayed by the long-term scheduler. Thus, this scheduler dictates what processes are to run on a system, and the degree of concurrency to be supported at any one time.

b. **Mid-term Scheduler**

The mid-term scheduler temporarily removes process from main memory and place them on secondary memory (such as a disk drive) or vice versa. This is commonly referred to as “swapping of processes out" or "swapping in" (also incorrectly as "paging out" or "paging in").

c. **Short-term Scheduler**

The short-term scheduler (also known as the CPU scheduler) decides which of the processes in the ready queue, in memory, are to be executed (allocated a CPU) next following a clock interrupt, an Input-Output (IO) interrupt and an OS call or another form of signal (Kadry*et al.,* 2011). Thus, the short-term scheduler makes scheduling decisions much more frequent than the long-term or mid-term schedulers. This scheduler can be preemptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process, or non pre-emptive (also known as "voluntary" or "co-operative"), in that case the scheduler is unable to force processes off the CPU.

2.4 CPU scheduling algorithms

The basic CPU scheduling algorithms are First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling (PS) and Round Robin (RR) (Achim, 2005). The FCFS is the simplest form of CPU scheduling algorithms which allocate CPU to the processes on the basis of their arrival to the ready queue.

Arriving processes are inserted in the tail (rear) of the ready queue and the process to be executed next is removed from the head (front) of the ready queue. A long CPU-bound process may dominate the CPU and may force shorter CPU bound processes to wait for a long period.

In the SJF, the scheduler arranged processes according to the shortest burst time in the ready queue, so that the process with least burst time is scheduled first (Achim, 2005). If two processes have equal burst time, the FCFS is applied. Long running processes may wait for prolonged periods, because the CPU has a steady supply of short processes. It has been proven to be the fastest scheduling algorithm, but it suffers from one important problem.

The PS associates each process with a priority number. The CPU is allocated to the process with the highest priority. If there are multiple processes with same priority, then FCFS will be used to allocate the CPU. Lower priority processes may starve, because the CPU may have a steady supply of higher priority process.

The Round Robin (RR) scheduling algorithm is designed specifically for time-sharing systems. It is a preemptive version of first-come, first-served scheduling. Processes are dispatched in a first-in-first-out sequence but each process is allowed to run for only a limited amount of time. This time interval is known as a time-slice or quantum. It is similar to FIFO scheduling but preemption is added to switches between processes. Typical quantum varies from 100 milliseconds to 2 seconds (Ishwari and Deepa, 2012).

The performance of RR scheduling is sensitive to time quantum selection, because if time quantum is very large then RR will be the same as the FCFS scheduling. If the time quantum is extremely too small then RR will be the same as Processor Sharing algorithm and number of context switches will be very high. Each value of time quantum will lead to a specific performance and will affect the algorithm's efficiency by affecting the processes waiting time, turnaround time, response time and number of context switches. This dissertation is concern in Round Robin Scheduling algorithm.

A good scheduling algorithm according to (Ajit*et al*; 2010) should possess the following characteristics:

a. Minimum context switches.

b. Maximum CPU utilization.

c. Maximum throughput.

d. Minimum turnaround time.

e. Minimum waiting time.

f. Minimum response time.

2.5 Process

A process is a program in execution. A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the programcounter and the contents of the processor's registers. A process generally also include the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables. A process may also include a heap, which is memory that is dynamically allocated during process run time (Silberschatz*et al*., 2013).

2.5.1 Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

a. **New**. The process is being created.

b. **Running**. Instructions are being executed.

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

d. **Ready**. The process is waiting to be assigned to a processor.

e. **Terminated**. The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems. However, certain operating systems also more finely delineate process states. It is important to realize that only one process can be runningon any processor at any instant. Many processes may be ready and waiting, however. The state diagram corresponding to these states is presented as thus:

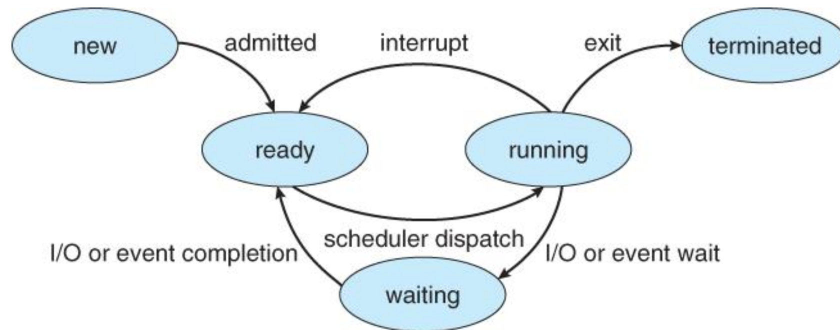

Figure 1: CPU Process State (Abdulrazaq*et al.* 2014)

2.6 Queuing Theory

A queuing system is described by its calling population, the nature of the arrivals, the service mechanism, the system capacity, and the queuing discipline. In the single-channel queue, the calling population is infinite; that is, if a unit leaves the calling population and joins the waiting line or enters service, there is no change in the arrival rate of other units that may need service (Jerry and John,2009). Arrivals for service occur one at a time in a random fashion; once they join the waiting line, they are eventually served. In addition, service times are of some random length according to a probability distribution which does not change over time. The system capacity has no limit, meaning that any number of units can wait in line.

Arrivals and services are defined by the distribution of the time between arrivals and the distribution of service times, respectively. For any simple single- or multi-channel queue, the overall effective arrival rate must be less than the total service rate, or the waiting line will grow without bound. When queues grow without bound, they are termed explosive or unstable networks in which units return a number of times to the same server. Before finally exiting the system, the condition about arrival rate being less than service rate may not guarantee stability (Jerry and John, 2009).

2.6.1 Characteristics of Queuing Model

The characteristics of queuing model according to (Kendall, 1953) are as follows:-

a. The arrival process of customers:

Usually, we assume that the interarrival times are independent and have a common distribution. In many practical situations, customers arrive according to a Poisson stream (i.e. exponential interarrival times). Customers may arrive one by one, or in batches. An example of batch arrivals is the customs office at the border where travel documents of bus passengers have to be checked.

b. The behaviour of customers:

Customers may be patient and willing to wait (for a long time), customers maybe impatient and leave after a while. For example, in call centers, customers willhang up when they have to wait too long before an operator is available, and theypossibly try again after a while.

c. The service times:

Usually, we assume that the service times are identically distributed,and that they are independent of the interarrival times. For example, the servicetimes can be deterministic or exponentially distributed. It can also occur that servicetimes are dependent of the queue length. For example, the processing rates of themachines in a production system can be increased once the number of jobs waitingto be processed becomes too large.

d. The service discipline:

Customers can be served one by one or in batches. We have many possibilities forthe order in which they enter service. The following were mentioned:

i. First come, first served, that is, in order of arrival

ii. Random order

iii. Last come,first served (e.g. in a computer stack or a shunt buffer in a productionline)

iv. Priorities (e.g. rush orders first, shortest processing time first)

v. Processor sharing (in computers that equally divide their processing power overall jobs in the system).

e. The service capacity

There may be a single server or a group of servers helping the customers.

f. The waiting room

There can be limitations with respect to the number of customers in the system. Forexample, in a data communication network, only finitely many cells can be bufferedin a switch. The determination of good buffer sizes is an important issue in the designof these networks.

2.6.2 Performance Measures in Queuing Model

Relevant performance measures in the analysis of queuing models according to (Kendall, 1953) are:

a. The distribution of the waiting time and the sojourn time of a customer. The sojourn time is the waiting time plus the service time.

b. The distribution of the number of customers in the system (including or excluding the one or those in service).

c. The distribution of the amount of work in the system. That is, the sum of service times of the waiting customers and the residual service time of the customers in service.

d. The distribution of the busy period of the server. This is, a period of time during which the server is working continuously.

2.7 CPU Time

The amount of time the CPU is actually executinginstructions is called CPU time. During the execution of most programs, the CPU sits idle much of the time while the computer fetches data from the keyboard or disk, or sends data to an output device. The CPU time of an executing program, therefore, is generally much less than the total execution time of the program. Multitaskingoperating systems take advantage of this by sharing the CPU among several programs.Most computers run synchronously utilizing a CPU clock running at a constant clock rate (frequency). Clock rate depends on the specific CPU organization (design) and hardware implementation technology used. A single machine instruction may take one or more CPU cycles to complete termed as the Cycles Per Instruction (CPI) (Shaaban, 2010).

For a specific program to run on a specific machine, CPU has the following parameters:

a. The total execution instruction count of the program (I)

b. The average number of cycle per instruction (CPI)

c. Clock cycle machine (C)

CPU execution time is the product of three parameters as follows:

$$CPU\ time\ =\ \frac{Seconds}{Program} = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

A program is comprised of a number of instructions executed(I) measured ininstructions/program. The average instruction executed takes a number of cycles per instruction (CPI) to be completed measured in cycles/instruction, CPI and CPU has a fixed clock cycle time C = 1/clock rate measured in: seconds/cycle

Therefore: $T = I \times CPI \times C$

Where:

T = Execution time per program in seconds

I = Number of instructions executed

CPI = Average CPI for program

C = CPU Clock Cycle (Shaban, 2010)

Example: A Program is running on a specific machine (CPU) withthe following parameters:

Total executed instruction count is 10,000,000 instructions, average CPI for the program is 2.5 cycles/instruction and CPU clock rates 200 MHz. (clock cycle = C = 5×10^{-9} secondsi.e in nanoseconds). What is the execution time for this program?

Since, CPU time = Instruction count x CPI x Clock cycle

CPU = 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x 5×10^{-9}

= 0.125 seconds

2.7.1 Instruction Count

Instruction is comprised of a number of elementary or micro operations which vary in number and complexity depending on the instruction and the exact CPU organization (Design). A micro operation is an elementary hardware operation that can be performed during one CPU clock cycle.This corresponds to one micro-instruction in microprogrammed CPUs.Examples of this operation are

a. Register operations such asshift, load, clear, increment

b. ALU operations: add, subtract, etc.

Thus, a single machine instruction may take one or more CPU cycles to complete termed as the Cycles Per Instruction (CPI)(Shaaban, 2010).

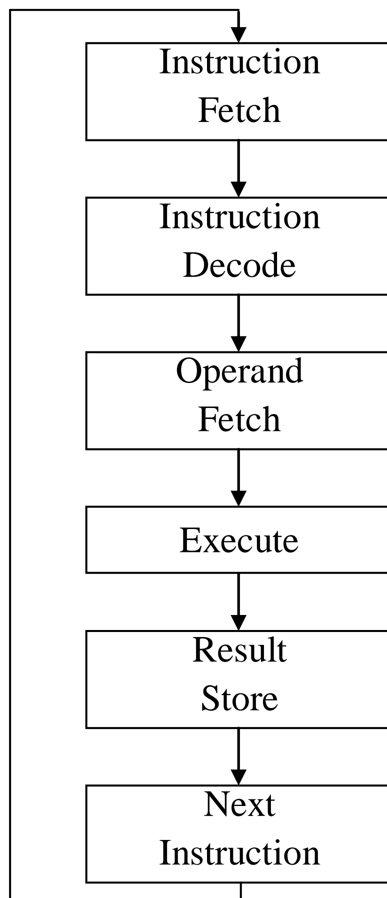

Figure 2.2: Generic CPU Machine Instruction Processing Steps (Shaaban, 2010)

2.7.2 Cycle per Instruction

Cycle per Instruction or Clock Per Instruction is the number of computer clock speed cycles alternating current pulses that occur while a computer instruction is being executed (Shaaban, 2010). The number of cycles per instruction can be reduced by using pipelining . In some superscalar processors, more than one instruction can be performed during a single clock cycle.

2.7.3 Clock Rate

The clock rate typically refers to the frequency at which a chip like CPU, one core of a multi-core processor, is running and is used as an indicator of the processor's speed. It is measured in clock cycles per second or its equivalent, the SI unit is hertz (Hz). The clock rate of the first generation of computers was measured in hertz or kilohertz (kHz), but in the 21st century, the speed of modern CPUs is commonly advertised in Gigahertz (GHz). This metric is most useful when comparing processors within the same family (Sarma*et al.* 2006).

2.8 Related Works

Review of works done by the various researchers in the field of Round Robin scheduling algorithm with assumption of CPU burst time, dynamic and fixed quantum time are considered in this session.

Sharma *et al.* (2010)proposed An Improved Round Robin Scheduling Algorithm for CPU scheduling. The work helps to improve the efficiency of Central Processing Unit (CPU) by allocating all processes to the CPU only one time like the Round Robin Scheduling Algorithm. After the first process is executed, it takes the next shortest job from the

processes in waiting queue and assign to the CPU. The next shortest job is also selected next, the process continues repeatedly until the all processes are executed. The algorithm combined the working of Shorted Job First (SJF) with contemporary Round Robin (RR) scheduling algorithm to improve the efficiency of CPU, process with large CPU burst will experience starvation.

Hamad*et al.* (2010)proposed an algorithm that uses integer programming to solve equations that decide a value that is neither too large nor too small such that every process has reasonable response time and the throughput of the system is not decreased due to un-necessarily context switches.The method depended on changing time quantum in each round over the cyclic queue called Changeable Time Quantum (CTQ) technique. The simulated results compared to Improved Round Robin by (Sharma, *et al.* 2010) and the traditional Round Robin algorithm in terms of waiting time, context switching and average turnaround time showed the algorithm outperformed the Round Robin. However, the result depended on the size of the pre-selected time quantum.

In Behera*et al.* (2010) A Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm (DQRRR) was presented. In this algorithm, number of context switching, average waiting time and average turnaround time was reduced. It does this, by arranging the processes in ascending order of their burst times present in the ready queue. Then, the time quantum is calculated. For finding an optimal time quantum, median method is used. Then, the time quantum is assigned to the processes. This time quantum is recalculated taking the remaining burst time in account after each cycle. In the next step, the algorithm have to re-arrange the sorted processes, i.e. among n processes, the process which needs

minimum CPU burst time will be replaced as the first process and then the process with highest CPU burst time from the queue, will be replaced as the second process and so on.

In Sahoo*et al.* (2011)New Fair-Share Scheduling with Weighted Time Slice was proposed and analyzed which calculates time quantum in each round. The time quantum is repeatedly adjustable according to the burst time of the currently running processes. Experimental analysis shows that their proposed algorithm gives better result, reduces the average waiting time, average turnaround time and number of context switches. The algorithm finds the time quantum in an intelligent way which gives better result than Changeable Time Quantum (CTQ). Weighted time slice method is used to get the optimal time quantum, where a weight (w) is assigned to each processes. Process having highest burst time is assigned the lowest say weight 1, process having next highest burst time is assigned weight 2 and so on. All the attributes like burst time, number of processes, weight of each process and the time quantum of all the processes are known before submitting the processes to the processor. All the processes are CPU bound.

Kadry*et al.*(2011) proposed New Round Robin Based Scheduling Algorithm for Operating Systems, Dynamic Quantum Using the Mean Average. The approach used is called AN algorithm to make the operating systems adjust the time quantum base on the burst of processes in the ready queue which solve the problem of fixed time quantum, increases the performance of round robin and the operating system finds the optimal time quantum without user's intervention. The determined time quantum represents real and optimal value because it is based on real burst time unlike the other methods, which depend on fixed time quantum value. Repeatedly, when a new process is loaded into the

ready queue in order to be executed, the operating system calculates the average of sum of the burst times of processes found in the ready queue including the new arrival process. This method needs two registers to be identified: SR: Register to store the sum of the remaining burst times in the ready queue. AR: Register to store the average of the burst times by dividing the value found in the SR by the count of processes found in the ready queue.When a process in execution finishes its time slice or its burst time, the ready queue and the registers will be updated to store the new data values. If this process finishes its burst time, then it will be removed from the ready queue. Otherwise, it will move to the end of the ready queue, then the SR will be updated by subtracting the time consumed by this process and AR will be updated according to the new data.

Sudhashree*et al.* (2011) proposed an algorithm called New Proposed Fittest Job First Dynamic Round Robin (FJFDRR) Scheduling Algorithm that compare its performance with Priority Based Static Round Robin (PBDRR) using fit factor and concept of dynamic time quantum. Fit factor is calculated and used to determine which process to be executed first. Two criteria are used to decide the execution of processes i.e higher user priority and shorter bust time. As user priority has higher importance than other factors, so it is given a weight age of 60% and burst time is given 40%, assuming that all the processes have same arrival time i.e. arrival time = 0, the User Priority = UP, User Priority Weight = UW, Shorter Burst time Priority = SP, Burst time Priority Weight = BW. Then Fit Factor "f" can be calculated as f = UP * UW + SP * BW. Attributes like burst time, number of processes and the user-priorities of all the processes are known before submitting the processes to the processor. All processes are CPU bound. No processes are I/O bound.

Sahu*et al.* (2012) developed Round Robin algorithm using Highest Response Ratio Next for Soft Real Time Systems. They aim at reducing the number of context switches, average waiting time and average turnaround time. The algorithm finds the dynamic time quantum by taking the mean of burst time of the processes and fills the Ready Queue according to arrival time. A Response Ratio is calculated for processes and the process with Highest Response Ratio is assigned the CPU. The Response Ratio is calculated as the summation of the remaining burst time is divided by the number of job in the queue. After the process is being assigned to the CPU again, the Response Ratio is calculated with the updated waiting time of the processes. This loop is continued until all the processes are being executed by the CPU. The processes with shorter burst time and higher waiting time are executed first resulting in better turnaround time and better waiting time. When burst time is increased, its works as first come first serve and also the overhead is high.

Ishwari and Deepa (2012) proposed Priority Based Round-Robin CPU Scheduling algorithm. It'sbased on the integration of round-robin and priority scheduling algorithm. It retains the advantage of round robin in reducing starvation and also integrates the advantage of priority scheduling. The algorithm performs by allocating CPU to every process in Round Robin fashion, according to the given priority, for a given time quantum (say k units) only for one time. After completion of the execution, the processes that still need the service of the CPU are arranged in increasing order of their remaining CPU burst times in the ready queue. New priorities are assigned according to the remaining CPU bursts of processes; the process with shortest remaining CPU burst is

assigned with highest priority. Then, the processes are executed according to the new priorities based on the remaining CPU bursts, and each process gets the control of the CPU until they finished their execution. They did not address starvation problem as with large time quantum, the algorithm tends to behave as First Come First Serve. However,Context Switching was still high and theiralgorithm assumes that all processes arrive at the same time in the ready queue which does not happen in real life.

Manish and Abdulkadir (2012) developed an algorithm to improved round robin CPU scheduling algorithm. The proposed improved Round Robin (IRR) CPU scheduling algorithm works similar to Round Robin (RR) with a small improvement. IRR picks the first process from the ready queue and allocate the CPU to it for a time interval of up to 1 time quantum. After completion of process's time quantum, it checks the remaining CPU burst time of the currently running process, if the remaining CPU burst time of the currently running process is less than 1 time quantum, the CPU again allocated to the currently running process for remaining CPU burst time. In this case this process will finish execution and it will be removed from the ready queue. The scheduler then proceeds to the next process in the ready queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than 1 time quantum, the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue. This work assumed that the environment where all the experiments are performed is a single processor environment and all the processes are independent. All the processes have equal priority. All the attributes like burst time, number of processes and the time slice of all the processes are known before submitting the processes to the

processor. The context switching time is equal to zero i.e. there is no context switch overhead incurred in switching from one process to another. All processes are CPU bound. No processes are I/O bound. The time quantum is taken in milliseconds.

Dhal *et al.* (2012) proposed algorithm that compares the performance analysis of Average Max Round Robin (AMRR) using Dynamic Time Quantum with Round Robin Scheduling Algorithm using Static Time quantum. The idea is to adjust the time quantum dynamically so that AMRR perform better than the Simple Round Robin. Burst time of processes are assumed and sorted in an increasing order so that it will give better turnaround time and waiting. The performance of round robin algorithm depends on size of fixed or static time quantum. In Round Robin algorithm the performance depends on the size of fixed Time Quantum (TQ). If TQ is too large then Round Robin algorithm approximate to First Come First Served (FCFS), if the TQ is too small, then there will be many context switching between the processes. Processes are already present in the Ready Queue (RQ) by default; Arrival Time (AT) is assigned to zero. The number of processes "n" and CPU Burst Time (BT) are accepted as input and Average Turnaround Time (ATT), Average Waiting Time (AWT) and number of Context Switch (CS) are produced as output. The formula below is used to generate the Time Quantum (TQ).

$$\text{AVG} = \frac{\sum Burst\ Time\ of\ all\ processes}{No\ of\ Processes} \qquad 2.1$$

$$\text{TQ} = \frac{AVG + MAXBT}{2} \qquad 2.2$$

Where AVG = Average

MAXBT = Maximum Burst Time

TQ = Time Quantum

Sharma and Dhakal (2012)a new proposed algorithm called Adaptive Round Robin Scheduling Algorithm is presented and it is based on Shortest Burst time using Smart Time Slice to schedule the processes in efficient and convenient way in order to reduce the turnaround time, response time and average waiting time for processes, the number of context switches and to reduce the load of CPU work. The number of processes is residing in the ready queue, assume their arrival time is assigned to zero and burst times are allocated to the CPU. The burst time and the number of processes (n) are accepted as input. Processes are arranged in increasing order according to their given burst time and chose smart time slice according to some conditions. The smart time slice will depends on the inputting number of processes of burst time.If numbers of processes vary, then smart time will be varied. The formula to calculate smart time is as thus: STS = average CPU burst time of all processes

Nutulapati and Bandarupalli (2012) introduced a new CPU algorithm called A Novel CPU Scheduling Algorithm which acts as both preemptive and non-preemptive based on the arrival time which helps to improve the CPU efficiency in real time uni-processor-multi programming operating system and also compare the result with the existing algorithm such as FCFS, SJF, Priority and Round Robin. The proposed algorithm introduced a factor called condition factor and is calculated by addition of burst time and the arrival time i.e Burst time + Arrival time. The calculated factors are assigned to each process and are arranged in ascending order. If the arrival time of first and second processes are equal the arranged based on their conditional factors then burst time is decrement and arrival time increment by 1, when burst time becomes zero then find the

waiting time and turnaround time of that process. Average waiting time is calculated by dividing total waiting time with total number of processes. Average turnaround time is calculated by dividing total turnaround time by total number of processes.

Adeli*et al.* (2012) proposed an algorithm used to predict the next CPU burst time called New Method of Adaptive CPU Scheduling using Fonseca and Fleming's Genetic Algorithm. The algorithm optimizes average waiting time and response time for the processes. CPU burst time defines how long they need CPU to complete the operation. Prediction of this time is, however, difficult for the operating system. Operation system will only make estimation of the time a process will take to execute. Prediction of next burst time presuming that the length of Next Bursts is similar to the length of previous bursts.It can be roughly calculated from the length of previous bursts. To do this the following formula is used: $T_{n+1} = \propto t_n + (1-\propto)T_n$ 2.3

Singla and Kanga (2013) Genetic Algorithm is developed to evaluate the performance of CPU Scheduling efficiently by using different operators. Two scheduling technique are used i.e round robin and multilevel queue for comparing the performance of this efficient genetic algorithm to minimize the average waiting time of processes to be executed. Genetic Algorithms are powerful and widely applicable stochastic search and optimization methods based on the concepts of natural selection and natural evaluation. Genetic algorithm are applied for those problems which either cannot be formulated in exact and accurate mathematical forms and may contain noisy or irregular data or it take so much time to solve or it is simply impossible to solve by the traditional computational methods. The Genetic Algorithm compares the performance of different scheduling

algorithms like round robin, multilevel queue and scheduling of these algorithms using genetic approach aim at minimizing the total waiting time.

Jain *et al.* (2013) An Improved CPU Scheduling Algorithm was proposed to improve the performance of CPU Scheduling by using a technique for increasing speed up factor known as "Pipelining". Pipelining is a technique in which a process is divided into sub operations and each sub operation is executed in a special dedicated segment that operates concurrently with all other segments. When CPU scheduler takes the decision of selecting the next process from the main memory, fetching and decoding of next process takes some time and this time latency can be avoided by using pipelining. To complete n processes, a k-segment pipeline requires $(k + (n-1))$ clock cycles. A non-pipeline unit will take $(n*Tn)$ time to complete n tasks where $Tn$is the time to complete each process.

Nirvikar and Kumar (2013), Performance Improvement Using CPU Scheduling Algorithm-SRT. The proposed algorithm improved all the drawback of existing round robin and presents a comparative analysis. It can be used in real time system, since it reduces the waiting time, throughput and turnaround time. Process with shortest job is selected and assigned to CPU for one time quantum (TQ). Following completion of one time quantum, it compares the time quantum with the remaining CPU burst time (RBT) if Remaining CPU Burst Time(RBT) is less than or equal to time quantum it assigns the same process again. However it continues repeatedly until all the processes have finished their execution. The environment in which the execution takes place is a single processor environment and all the processes are independent.

Farooqui and Shoaib(2014) developed a comparative review of CPU scheduling algorithms to summarize major CPU scheduling algorithms proposed till date and evaluate their performance. The comparative review is performed on different algorithm such as First Come, First Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Round Robin (RR), Priority Scheduling (PS), Highest Response Ratio Next (HRRN) and Longest Job First (LJF). The analysis comparison over a set of data clearly depicts that Shortest Remaining Time First (SRTF) takes minimal time to complete execution of processes and have approximately minimal average waiting time while Longest Job First (LJF) appear to be slow in performance.

Ragha*et al.*(2014) developed a Dynamic Quantum Based Genetic Round Robin Algorithm to improve the performance of time quantum dynamically for all iterations instead of a static quantum using genetic approach. Genetic algorithm is an evolutionary technique which finds optimal solutions for New Population (NP) hard problems. The workpresent comparison of three algorithms: Round Robin with static quantum, Round Robin with dynamic quantum and Genetic algorithm based Round Robin with dynamic quantum. Average waiting time is selected as a fitness of the chromosome. The fittest chromosome is one with minimum average waiting time. Round Robin algorithm with dynamic quantum is used to calculate fitness of the chromosome. Initially, the quantum is the median of all the processes burst time. After first iteration, the remaining burst times of processes are arranged so that the median is selected among them, the procedure continues until all processes are executed.

Sonagara (2014) proposedRound Robin CPU Scheduling Using Dynamic Time Quantum with Multiple Queue which is achieved by comparing different multilevel queue and round robin scheduling algorithm which contain different method of finding time quantum. Round robin CPU scheduling using dynamic time quantum with multiple queue gives better output and is efficient compared to round robin algorithm. In this technique, first find out the median and mean value of CPU Burst time of all the processes .then we compare mean and median value. Among them, greater value multiply with highest burst time and least value multiply with lowest burst time and then we find square root of it. This value taken as a Dynamic time quantum and apply to each processes.

Mittaland Raman(2014) proposed an Efficient Dynamic Round Robin CPU Scheduling Algorithm (EDRR)for timeshared systems which help to overcome the disadvantage ofround robin such as starvation, waiting time, turnaround time and the throughput. Comparative analyses were carried out between few scheduling algorithm to show the efficient and performance of the proposed algorithm. Processes are arranged in increasing order of their CPU burst time. Set the time quantum is equal to the CPU burst time of the first process (The shortest process). Calculate the median and mean of CPU burst time of all the processes. Set time quantum (TQ) according to following method if (mean>median) then TQ = ceil (sqrt((mean * highest burst time) +(median * lowest burst time))) Else If (median>mean) TQ = ceil (sqrt((median * highest burst time) +(mean * lowest burst time))) Else TQ = mean. The algorithm is the combination of the shortest job first CPU scheduling algorithm and the round robin CPU scheduling algorithm with efficient and dynamic time quantum.

Ashiruet al. (2014a) proposed an algorithm called Dynamic Round Robin with Controlled Preemption (DRRCP). In this algorithm any process that has achieved 95% of its burst will run to completion otherwise it will be preempted. Although Ashiru et al. (2014) achieved minimum response time but other performance evaluation were very high compare to other improved versions.

Ashiruet al.(2014b) also proposed an algorithm named Half Life Variable Quantum Time Round Robin (HLVQTRR) which has two phases. In the first phase, it ensures that every process execute by half of its burst time. While in the second phase, every process run to completion. In their algorithm they were able to achieved minimum response time but the algorithm is not justifiable considering the fairness of Round Robin since processes have different burst time, process with high burst tends to run for a long period of time compare to the one with low burst time. Average Waiting Time and Average Turnaround Time is very high compare to other improved versions.

Abdulrazak*et al.* (2014)proposed an algorithm that improved the performance of New Improved Round Robin CPU Scheduling Algorithm, implemented and benchmarked against five other algorithms. The proposed algorithm compared with other algorithms produces minimal average waiting time (AWT), average turnaround time (ATAT) and number of context switches (NCS). The work assumes another queue called the ARRIVE queue which holds processes according to their arrival times while there are other processes in the ready queue (say REQUEST) waiting for CPU allocation.Quantum time was determined by calculating the average burst time.

Ghani*et al.*(2015) developed an algorithm, Priority Algorithm with Dynamic Quantum Time (PDQT)which focuses on the pitfall of RR by improving the concept of improved round robin with varying time quantum (IRRVQ). Changing the time quantum of each round to reduce context switch, waiting time, turnaround time, less starvation and also obtained higher throughput. Priorities and quantum times are assigned according to the CPU bursts of processes; the processes with lowest burst time is set with highest priority, choose the first process in Round-Robin fashion and assign its burst time as quantum time and allocates CPU to this process only for one quantum time. Time quantum is calculated using this formula: $q=k+p-1$, where q is the new quantum time, k is the old quantum time, and p is the priority of the processes in the ready queue

Aisha (2016) proposed an algorithm to improved Longest Job First CPU Scheduling Algorithm by introducing a new method of determining the threshold that will be used in the categorization of the processes and merging shorter processes. The algorithm arranges all processes in the ready queue in descending order of their burst times. It uses the principle of percentile to determine the threshold for categorizing the processes.

Faiza (2016) proposed an algorithm, design and analysis of improved round robin algorithms with arrival time to modified an algorithm by Ashiru (2014), two algorithms Dynamic Quantum with Readjusted Round Robin and Half Life Variable Quantum Time Round Robin were modified by considering arrive time to be random instead of static. The algorithm compared with that of Ashiru (2014) performs better in term of AWT, ATAT and NCS. The algorithm did not address the issue of burst time, the burst time is assumed to be known.

Anju*et al.*(2016)proposed a variant of RR algorithm called Dynamic Time Slice Round Robin Scheduling Algorithm with Unknown Burst Time. The approach was tuning the quantum at run time. Initial value was assignedas time quantum and run through the first cycle. If no process finish in first cycle, the time quantum is multiplied by two, if the number of processes that finish in the first cycle is greater than two, it divides the time quantum by two, otherwise if at least one is completed, it continues without changing the initial time quantum. Their algorithmfind 15% reduction in average waiting time,15% in average turnaround timeand number of context switches was also reduced by 10%. This dissertation proposal therefore, will modify this algorithm.

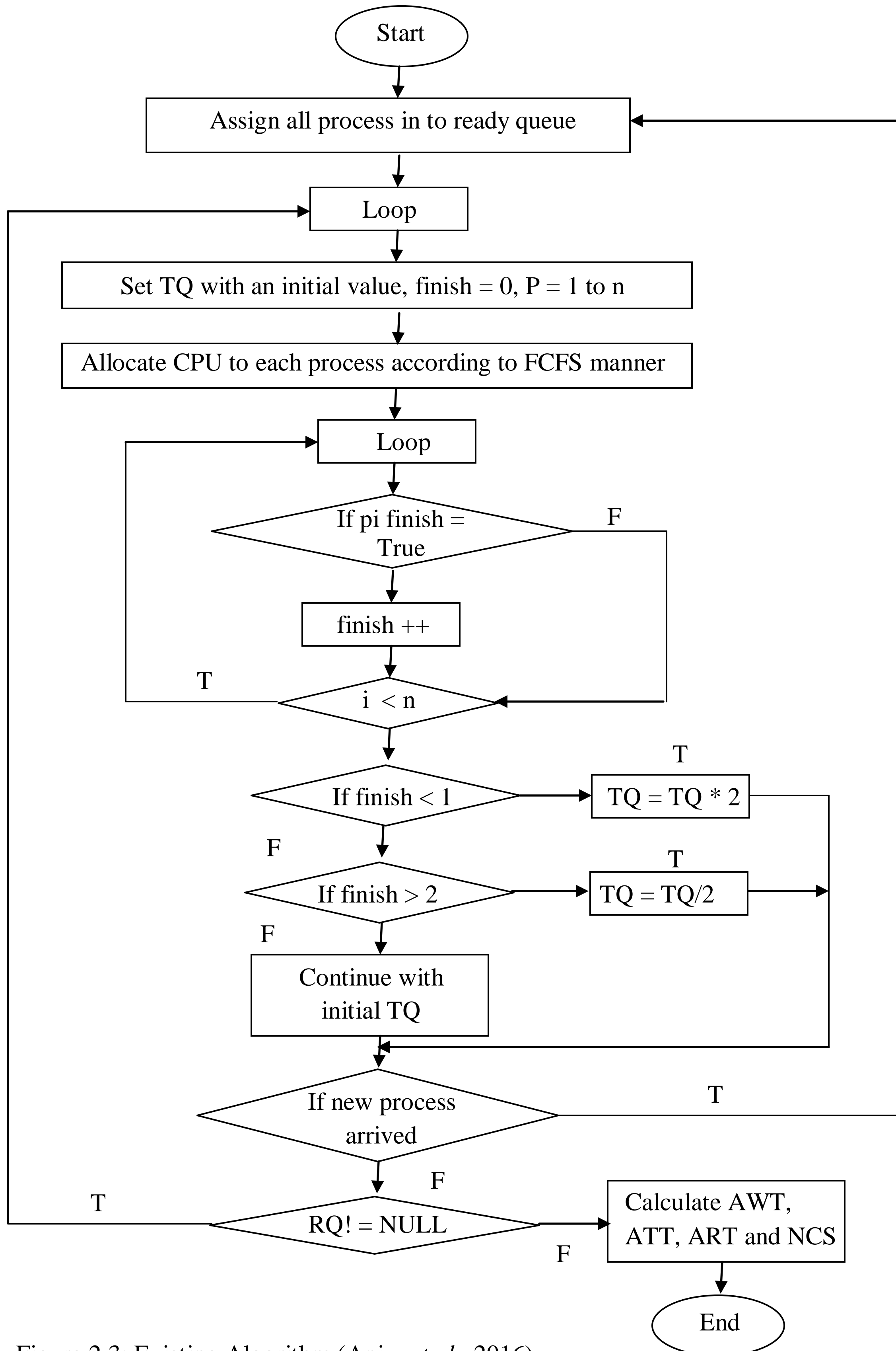

Figure 2.3: Existing Algorithm (Anju, *et al.,* 2016)

First and foremost, all processes are assigned to ready queue, a small value is assigned as initial time quantum, and execution is carryout through the first cycle with the initial time quantum.CPU is allocated to each process according to FCFS manner. Initial value of finish is zero (0) before execution, i is index of process and n is the total number processes in the ready queue. If Pi finish is true, finish is plus 1, if finish is less than 1, they multiply the time quantum by two if finish is greater than 2, they divide time quantum by 2 else continue with the initial time quantum. However, if new process arrived, it will be added to the ready queue and if no process in the ready queue, it calculate Average Waiting Time, Average Turn Around Time, Average Response Time and Number of Context Switches.

2.9 Literature Gap

Several approaches have been used for CPU scheduling algorithm, from the reviewed literatures, it was observed that their concentration is mostly on quantum time, some used fixedquantum time while others used dynamic quantum time and parameters are known before submitting the process.Dynamic Time Slice Round Robin with Unknown Burst Time by Anju*et al*. (2016) proposed an algorithm to determine burst time using initial time quantumyet there is a scope to design and implement a scheduling algorithm that can enhance CPU utilization by using instruction count to predict burst time, random arrival time and dynamic quantum time to further improve the efficient performance of CPU scheduling in real time environment. However, this work will address the limitation of the reviewed worksuch asAnju*et al.* (2016), Manish and Faizu(2014) with problem of assumptions of parameter before scheduling.

CHAPTER THREE

DESIGN OF IMPROVED DYNAMIC TIME SLICE ROUND ROBIN SCHEDULING ALGORITHM WITH UNKNOWN BURST TIME

3.1 Introduction

This chapter presents the proposed Improved Dynamic Time Slice Round Robin with Unknown Burst Time, its Pseudo code and flowchart show how it works with the aid of an illustrative example.

3.2 The Proposed Dynamic Time Slice Round Robin with Unknown Burst Time (DTSRRUBT) Scheduling Algorithm

The proposed IDTSRRUBT CPUscheduling algorithm is the modification of Dynamic Time Slice Round Robin with Unknown Burst Time (DTSRRUBT) CPU scheduling algorithm (that is an algorithm by Anju*et al*, (2016)). The algorithm introduced a technique that calculates burst time using instruction count, process arrives randomly and quantum time is dynamically determined.

A model that calculate burst time of processes is designed, number of processes to be executed are required to be entered with the frequency while burst times are generated for those processes and their details. Burst time is calculated as thus:

For example a 900MHz processor was used to execute a benchmark program with the following six instruction type and clock cycle count.

Table 3.1: Instruction Type

S/N	Instruction Type	Instruction Count	Clock Cycle Count
1.	Data Transfer	1000	3
2.	Instructions Fetch	1500	2
3.	Branch Instructions	4000	2
4.	Floating Point	4500	2
5.	Input/output Fetch	8000	1
6.	Store Instructions	2500	2

$\sum instruction\ count: 21500$

$$CPI = \frac{\sum(IC) \times (CCI)}{IC} \quad 3.1$$

Where CPI = ClockPer Instruction

IC = Instruction Count

CCI = Clock Cycle Per Instruction

$$CPI = \frac{1000 \times 3 + 1500 \times 2 + 4000 \times 2 + 4500 \ \times 2 + 8000 \times 1 + 2500 \times 2}{21500}$$

$$CPI = 1.67$$

Therefore:

$$Execution\ time\ (T) = \ CPI \ \times Instruction\ Count \ \times Clock\ time = \frac{CPI \times Instruction \ \ Count}{frequency} \quad 3.2$$

$$(T) \ = \frac{1.67 * 21500}{900 * 21500} \frac{1.67}{900} \quad = \quad 0.018556$$

Therefore:

0.0018556 * 1000

Burst Time: = 1.86

However, after generating the burst time for the processes and arrival time, the algorithm takes the first process and assigns to CPU for initial time quantum of three (3), while executing the first process, it checks if other process has arrived the ready queue, if processes arrived it calculates average burst time of the processes as time quantum, each process then runs for the time quantum calculated, process that could not finish execution moves to the tail of the ready queue. However, it continues in the same fashion until all processes finished execution and if no process again in the ready queue, it calculates average waiting time, average turnaround time, average response time and number of context switch.

3.2.1 The pseudo code of proposedDynamic Time Slice Round Robin CPU Scheduling Algorithm with Unknown Burst Time

Step 1: Start

Step 2: Enter the number of processes to be processed

Step 3: Enter the frequency of the processor

Step 3: System generates the burst time for processes with random arrival time

Step 4: WHILE (READY QUEUE!= NULL)

Step 5: If ($process_Index == 1$)

$time_{quantum} = current\ time\ quantum$

Assign the first process ($pr[process_Index]$) to CPU

Else

$$time_quantum = \left\lceil \frac{\sum_{i=1}^{n} burst\ \ time\ [i]}{n} \right\rceil$$

Step 5: END If

Step 6: Allocate the CPU to the first process in ARRIVE queue for a period of 1 time quantum.

Step 7: If the burst time of the currently running process is remaining it moves the process to the tail of the arrive queue and if it finishes it removes the process from queue and go to step 4.

Step 8: If a new process arrive the system, it is placed in the ARRIVE queue.

Step 9: END WHILE.

Step 10: Calculate AWT, ATAT, ART and NCS.

Step 11: END

3.2.2 The flow chart

Figure 3.1 shows the flow chart of the proposed Dynamic Time Slice Round Robin CPU scheduling algorithm, the shaded figures are the improvement made.

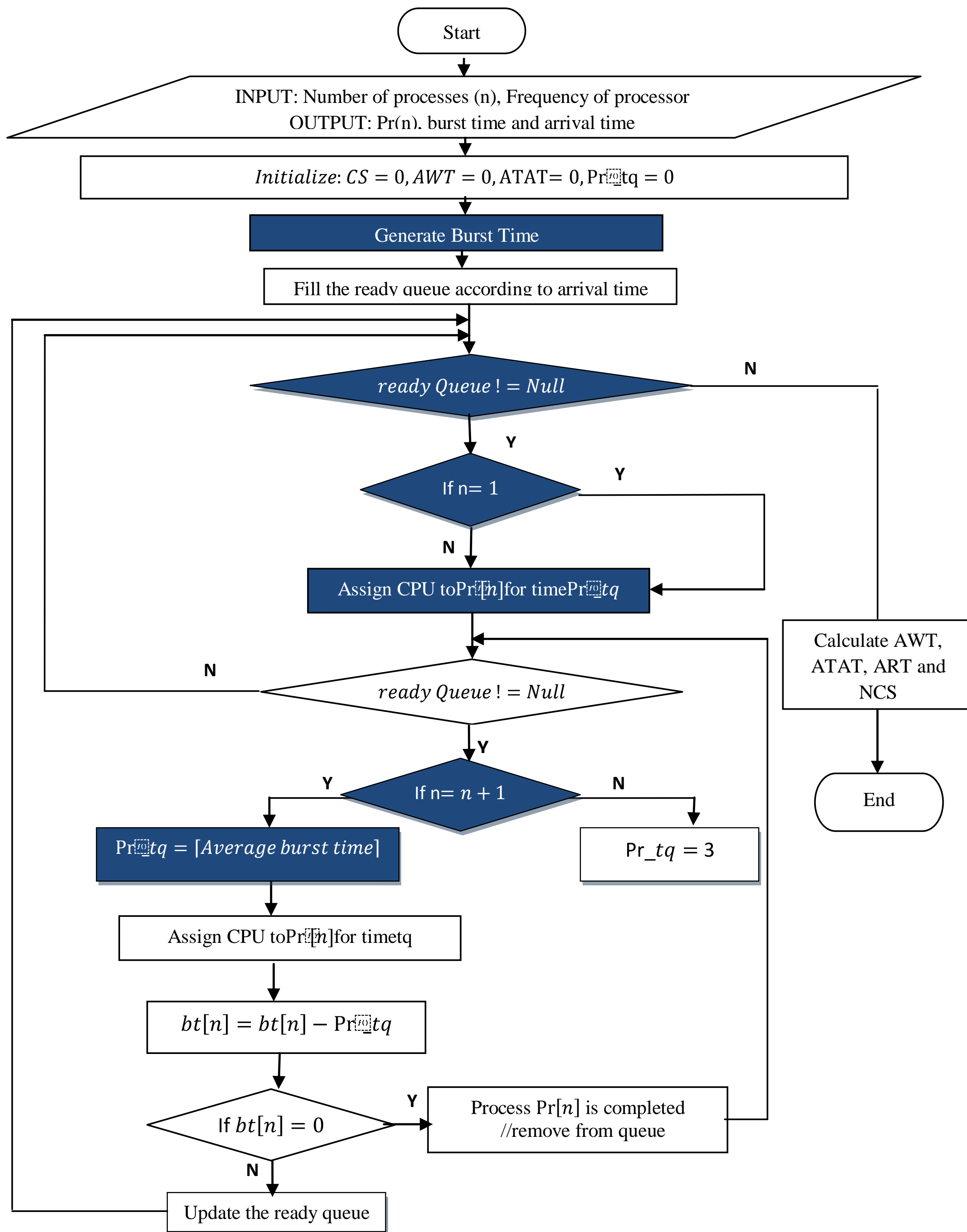

Figure 3.1: The Flow Chart of the Proposed Dynamic Time Slice Round Robin CPU Scheduling Algorithm

3.3 How the Proposed Algorithm Works

Following all steps of the proposed algorithm stated in the sections 3.2.1, the example that follows will illustrate how the algorithm works. Given 15 processes with their arrival time and burst times as shown in Table 3.1, the proposed algorithm works as follows:

The algorithm executes the first job while subsequent processes are addedto the ready queue. New time quantum will be generated by calculating the average burst time of the arrived queue. While the CPU is processing the jobs in the Ready Queue, Quantum time is added to the Time in the CPU so the program re-check if any process has arrived and add to the Ready Queue and generate a new Quantum time until no process is available again in the ready queue.

Table 3.2: Illustrative Table

Process ID	Burst Time	Arrival Time
P1	5	0
P2	7	2
P3	8	3
P4	5	4
P5	7	6
P6	4	7
P7	3	9
P8	7	13
P9	6	16
P10	5	17
P11	4	20
P12	8	21
P13	6	21
P14	4	25
P15	3	27

The first process will execute for initial time quantum, Current-Time=3, running =true, at Current-Time=3, P2, P3, and P4 have arrived, Quantum=(7+8+5)/3=6.7 while Current-Time+=Quantum;at Current-Time=9.7, P5, P6, and P7 have Arriveda new Quantum will be generated that isP3, P4, P5, P6, and P7 but P2 is out of the loop because it has finished executing.Quantum=(8+5+7+4+3)/5=5.4;Current-Time+=Quantumat Current-Time=15.1P8 and P9 have arriveda new Quantum will be generated that is P3, P4, P5, P6, P7, P8, P9and time quantum will be (8+5+7+4+3+7+6)/7Quantum=5.7;Current-Time+=Quantumat Current-Time=20.8P10, P11, P12, and P13 while new Quantum will be generated thus:P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13Quantum=(8+5+7+4+3+7+6+5+4+8+6)/11=5.7Current-Time+=Quantum;at Current-Time=26.5P14 and P15 have arriveda new Quantum will be generatedagainP3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, and P15Quantum=(8+5+7+4+3+7+6+5+4+8+6+4+3)/13=5.4

CHAPTER FOUR

IMPLEMENTATION, RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter presents the design and implementation of the modified Dynamic Time Slice Round Robin with Unknown Burst Time CPU scheduling algorithm. It starts by displaying a model that generates burst time of processes with their arrival time along the details of each process. However, simulation of the various scheduling algorithms under study will be illustrated in table format and graph. Finally, the results of this implementation are analyzed.

4.2 Implementation

Performance criteria, that is, average waiting time, average turnaround time, average response time and numbers of context switches were studied. First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), Improved Round Robin (IRR), Dynamic Time Slice Round Robin with Unknown Burst Time and the proposed Improved Dynamic Time Slice Round Robin with Unknown Burst Time CPU scheduling algorithm were simulated to observe these criteria. The simulation environment where all the experiments were performed was a single processor environment and all the processes are independent and CPU bound, no process was I/O bound.

To demonstrate the previous considerations using five processeswith 1024MHz frequency, the following example will be considered, in which each process has its burst time and arrival time as shown in Tables 4.1. Figure 4.1 is the interface where the

number of processes and frequency is inputted. However, table 4.2 to table 4.7 show the performance analysis of the six algorithms and figure 4.2 to figure 4.7 show the graph of the six algorithms. Time slice for RR and IRRis fixed while for DTSRRUBT and IDTSRRUBT will be dynamic, all processes arrived randomly.

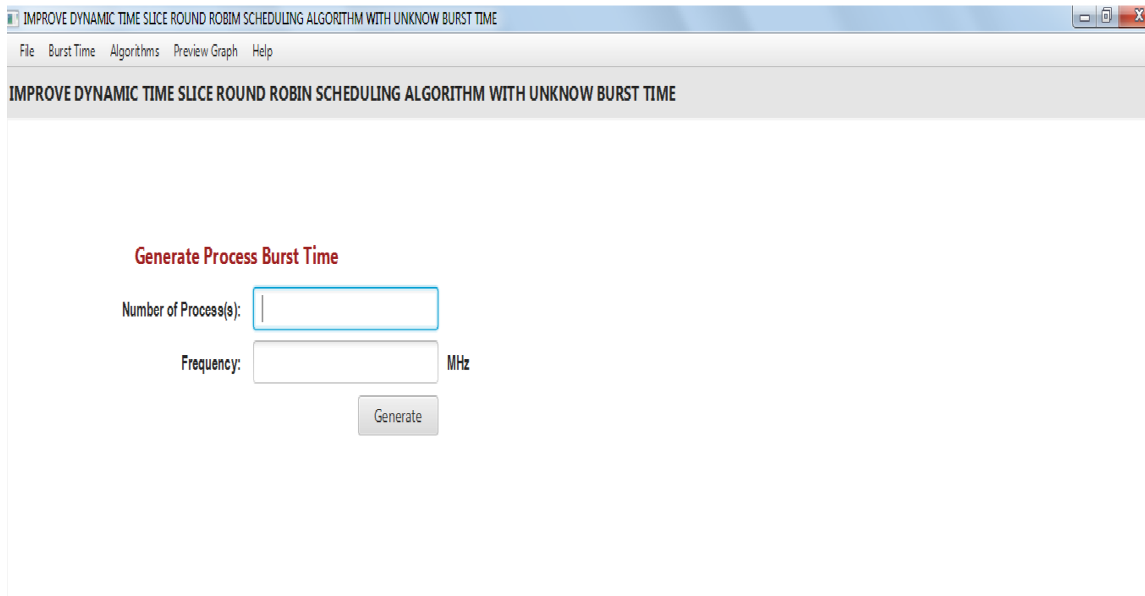

Figure 4.1: IDTSRR Interface

Table 4.1: Process Details

Process ID	Arrival time (ms)	Burst time (ms)
P1	0.0	3.2
P2	2.0	2.87
P3	2.0	2.86
P4	3.0	3.61
P5	5.0	3.45

Table 4.2: First Come First Serve

Process ID	Arrival Time	Burst time	Start time	Finish Time	Waiting Time	Turnaround Time	Response Time	Context Switch
P1	0.0	3.2	0.0	3.2	0.0	3.2	0.0	1
P2	2.0	2.87	3.2	6.07	1.2	4.07	1.2	1
P3	2.0	2.86	6.07	8.93	4.07	6.93	4.07	1
P4	3.0	6.61	8.93	12.54	5.93	9.54	5.93	1
P5	5.0	3.45	12.54	15.99	7.54	10.99	7.54	1
Average					3.75	6.95	3.75	5

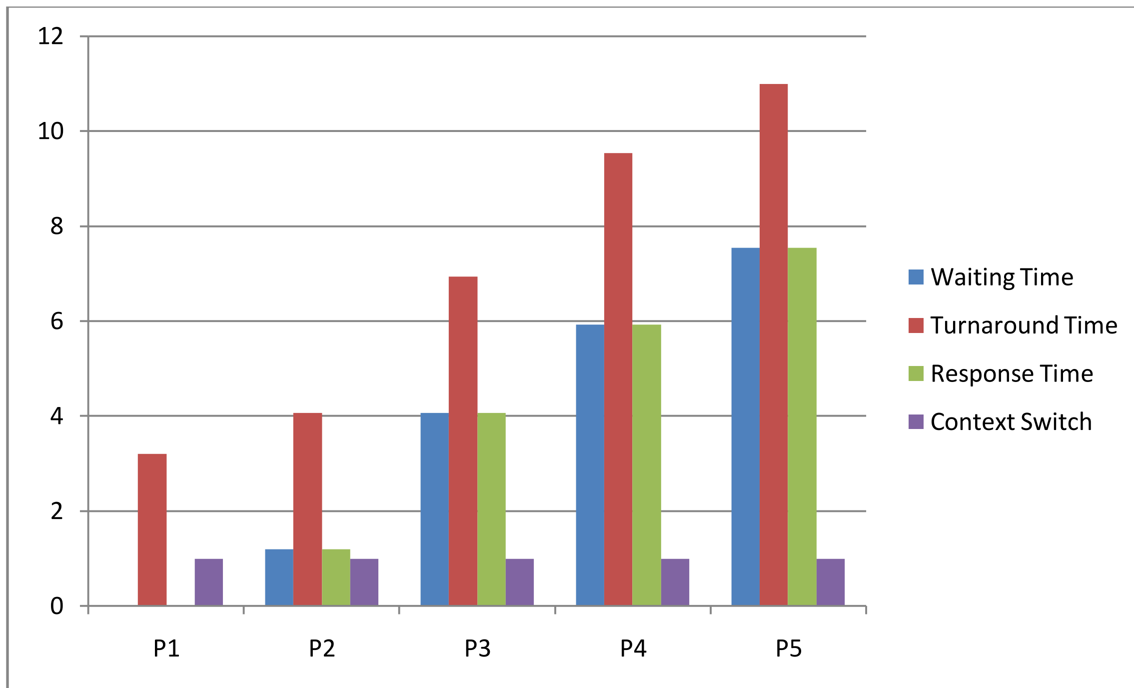

Figure 4.2: FCFS Graph

Table 4.3: Shortest Job First

Process ID	Arrival Time	Burst time	Start time	Finish Time	Waiting Time	Turnaround Time	Response Time	Context Switch
P1	0.0	3.2	0.0	3.2	0.0	3.2	0.0	1
P3	2.0	2.86	3.2	6.06	1.2	4.06	1.2	1
P2	2.2	2.87	6.06	8.93	4.06	6.93	4.06	1
P5	5.0	3.45	8.93	12.38	3.93	7.38	3.93	1
P4	3.0	3.61	12.38	15.99	9.38	12.99	9.38	1
Average					3.71	6.27	3.71	5

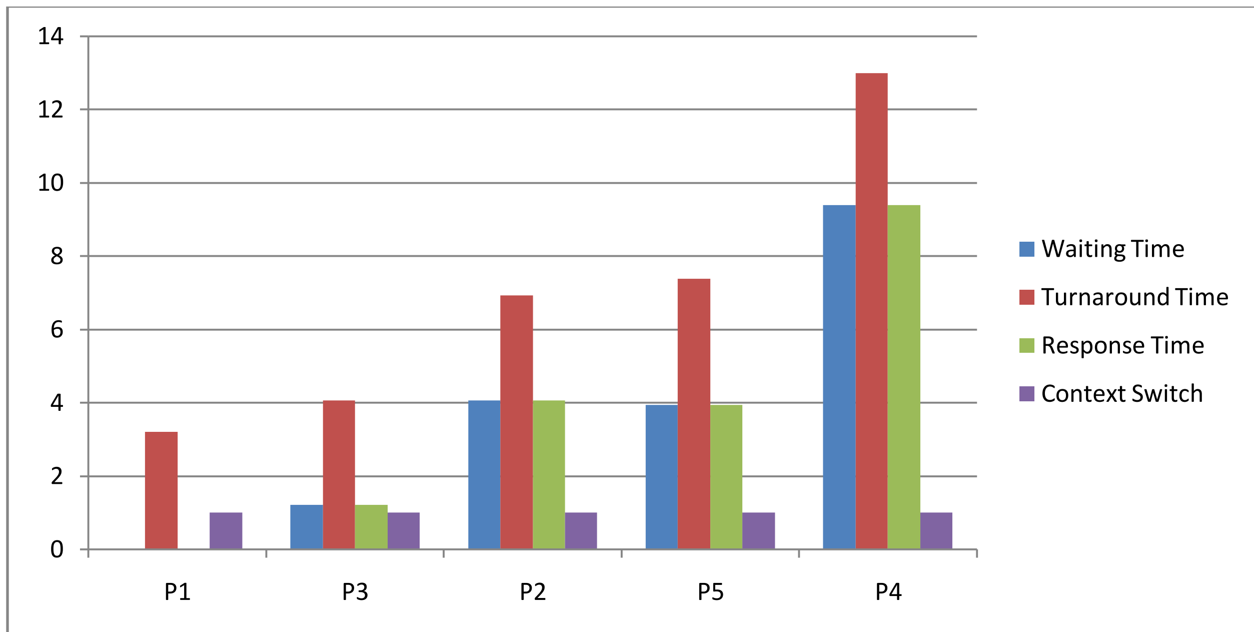

Figure 4.3: SJF Graph

Table 4.4: Round Robin

Process ID	Arrival Time	Burst time	Start time	Finish Time	Waiting Time	Turnaround Time	Response Time	Context Switch
P1	0.0	3.2	0.0	7.2	4.0	7.2	0.0	2
P2	2.0	2.87	2.0	10.07	5.2	8.07	0.0	2
P3	2.0	2.86	4.0	12.93	8.07	10.93	2.0	2
P4	3.0	3.61	7.2	14.54	7.93	11.54	4.2	2
P5	5.0	3.45	10.07	15.99	7.54	10.99	5.07	2
Average					6.55	9.75	2.25	10

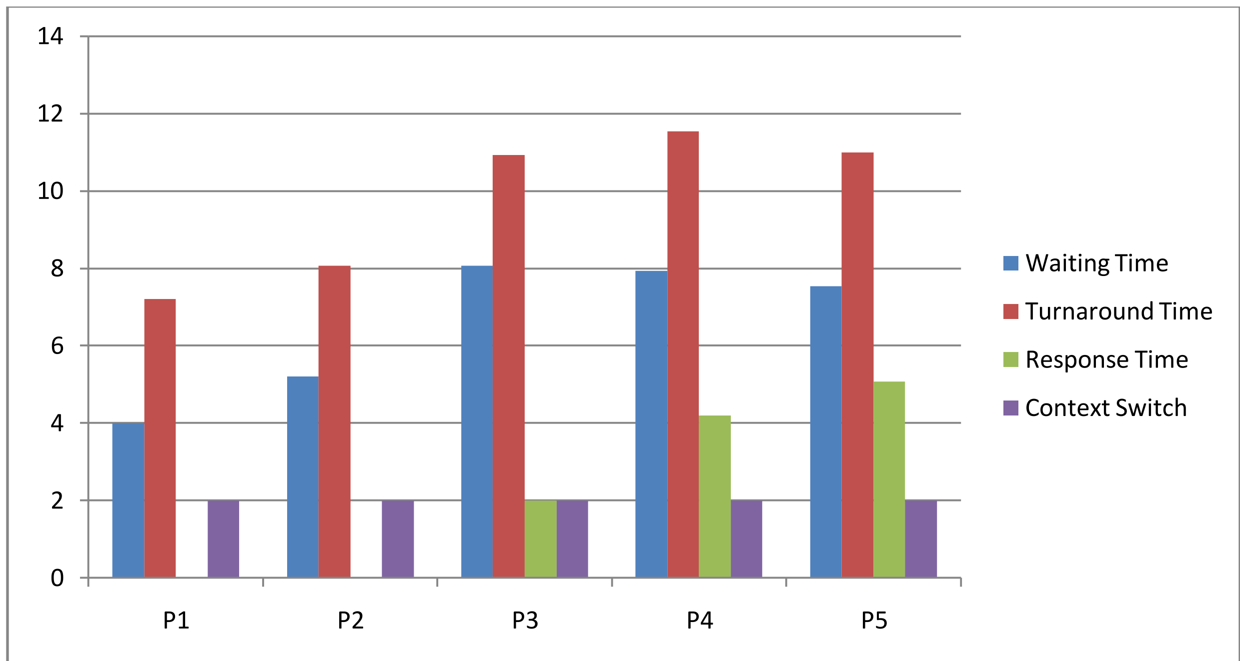

Figure 4.4: Round Robin Graph

Table 4.5: Improved Round Robin

Process ID	Arrival Time	Burst time	Start time	Finish Time	Waiting Time	Turnaround Time	Response Time	Context Switch
P1	0.0	3.2	0.0	3.2	0.0	3.2	0.0	1
P2	2.0	2.87	3.2	6.07	1.2	4.07	1.2	1
P3	2.0	2.86	6.07	8.93	4.07	6.93	4.07	1
P4	3.0	3.61	8.93	15.54	8.93	12.54	5.93	2
P5	5.0	3.45	11.93	15.99	7.54	10.99	6.93	2
Average					4.35	7.55	3.63	7

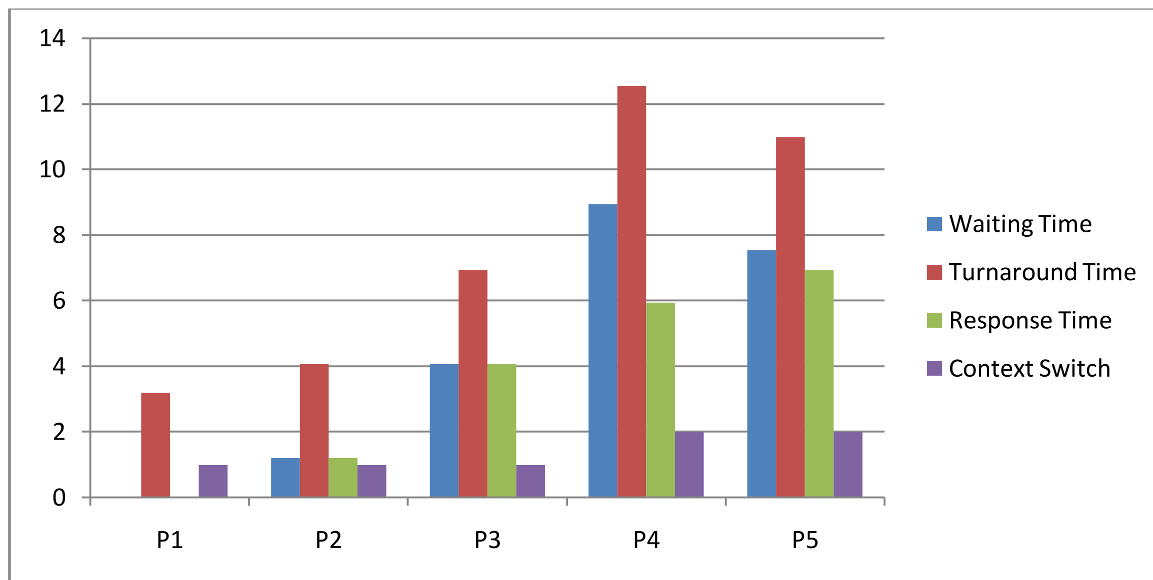

Figure 4.5: Improved Round Robin Graph

Table 4.6: Dynamic Time Slice Round Robin

Process ID	Arrival Time	Burst time	Start time	Finish Time	Waiting Time	Turnaround Time	Response Time	Context Switch
P1	0.0	3.2	0.0	7.7	4.5	7.7	0.0	2
P2	2.0	2.87	2.0	10.57	5.7	8.57	0.0	2
P3	2.0	2.86	4.5	13.43	8.57	11.43	2.5	2
P4	3.0	3.61	7.7	14.54	7.93	11.54	4.7	2
P5	5.0	3.45	10.57	15.49	7.04	10.49	5.57	2
Average					6.75	9.95	2.55	10

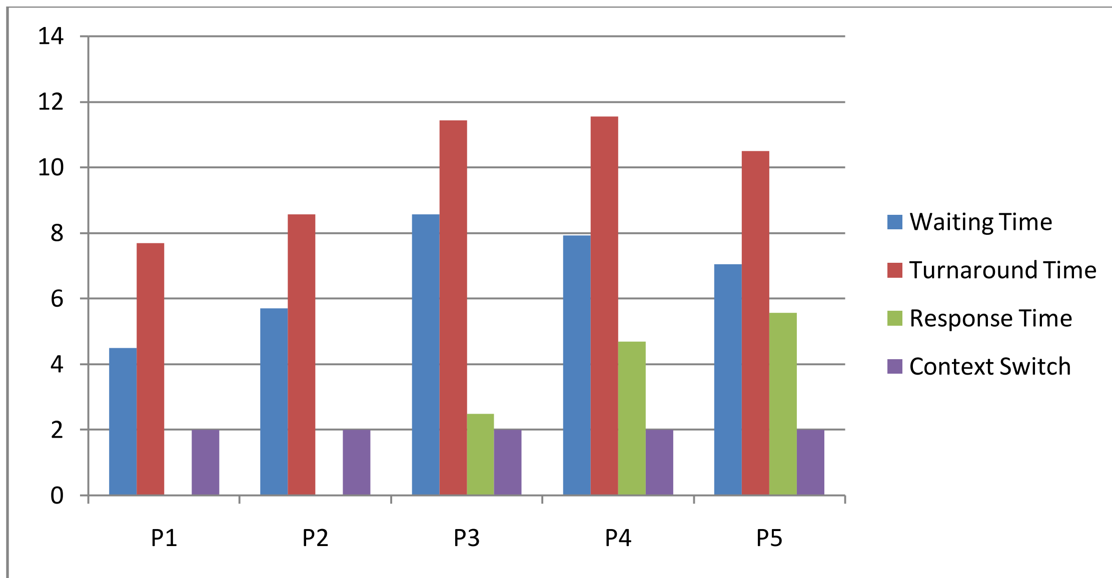

Figure 4.6: DTSRR Graph

Table 4.7: Improved Dynamic Time Slice Round Robin

Process ID	Arrival Time	Burst time	Start time	Finish Time	Waiting Time	Turnaround Time	Response Time	Context Switch
P1	0.0	3.2	0.0	12.54	9.34	12.54	0.0	2
P2	2.0	2.87	3.0	5.87	1.0	3.87	1.0	1
P3	2.0	2.86	5.87	8.73	3.87	6.73	3.87	1
P4	3.0	3.61	8.73	12.34	5.73	9.34	5.73	1
P5	5.0	3.45	12.54	15.99	7.54	10.99	7.54	1
Average					5.5	8.69	3.63	6

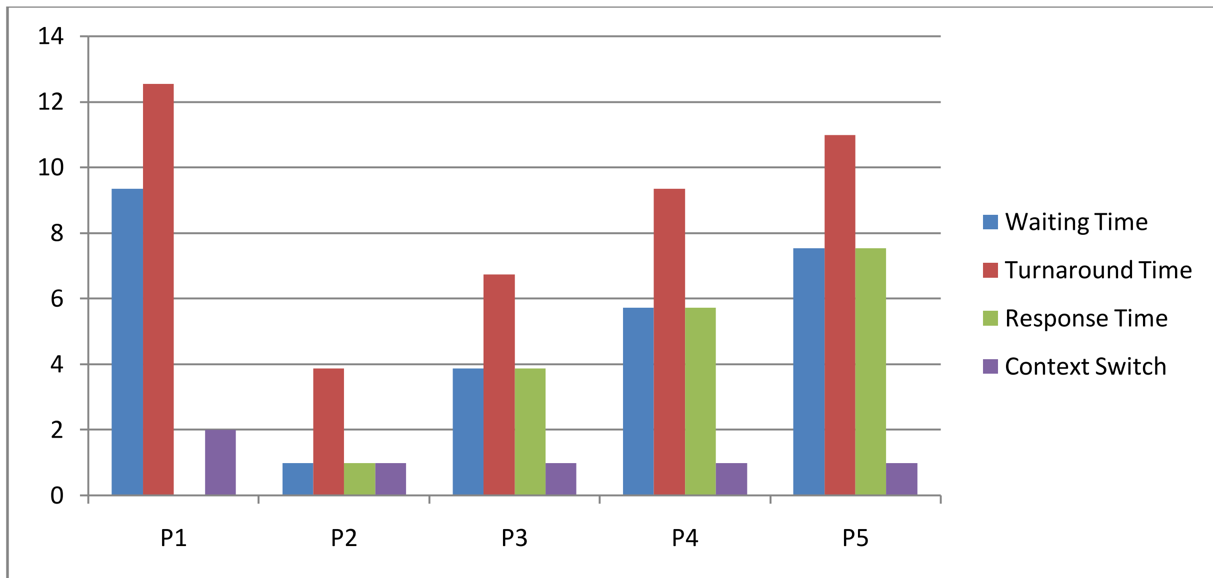

Figure 4.7: Improved Dynamic Time Slice Round Robin Graph

Table 4.2 shows simulation results for FCFS algorithm and Fig.4.2 shows the graph of the result from the table, from the graph the response time and the waiting time is high but the context switch is low since process switch once.Table 4.3 shows simulation results for SJF algorithm and Fig. 4.3 shows the graph of the result from the table, the

performance is better compare to FSCS in term of AWT and ART since priority is given to process with short burst time. Table 4.4 shows simulation results for conventional RR along with the graph in Fig. 4.4. Table 4.5 shows simulation results for IRR along with the graph in Fig. 4.5. Table 4.6 is the results of simulation for DTSRR and Fig. 4.6 shows the graph of DTSRR and Table 4.7 is the results of proposed algorithm simulated along with the graph in Fig. 4.7. However,in the performance of the three round robin algorithm when compared, IDTSRR is better. In the graph, X-axis represents the time and Y-axis represents the process ID.

4.3 System Requirement

4.3.1 Experimental setup

Hardware

a. Hewlett Packard (HP) Compaq Presario CQ61 Notebook PC with a T4300 processor running at 2.10GHz

b. 3.00GB RAM and

c. 220GB of hard disk

Software

a. Window 7 Home Basic

b. NetBeans IDE 8.01 version and JCreator LE

Each process in the process set is a tuple: < (process_*id*, *Arrival time and Burst time*)>.

A process burst time generator was developed to take care unknown burst time of different processes in the system. Burst time (i.e. the *CPU_time*) was generated using instruction count.

4.4 System Architecture

The logical architecture of the system is depicted in Figure 4.11. The system takes in as input the number of processes to be considered and the frequency of the processor, instruction count of each process is used to generate burst time of each process with arrival time randomly generated. The generated processes are stored in the process log and are arranged in the ready queue as the scheduling algorithm requires. The CPU is allocated to the processes by using each of the scheduling algorithms in the scheduler module and the performance criteria of each of the algorithms are evaluated and stored in the criteria log. The evaluated criteria are also displayed on the computer screen and the performance graph of each of the simulation for all the algorithms.

Figure 4.11: System Architecture

4.5 Evaluation of Performance

Evaluation of performance is based on the following criteria:

a. Waiting Time (WT): The amount of time before a process starts after first entering the ready queue (or the sum of the amount of time a process has spent waiting in the arrive queue). The formula to calculate waiting time is

$$WT = Time\ of\ first\ scheduled - Arrival\ time \quad 4.1$$

AWT = Average Waiting Time

$$AWT = \frac{sum\ of\ all\ processes\ waiting\ time}{number\ of\ processes} \quad 4.2$$

b. Turnaround Time (TAT): Amount of time to execute a particular process from the time of submission through the time of completion. The formula to calculate turnaround time is as thus:

$$TAT = Time\ of\ process\ completion - Arrival\ time \quad 4.3$$

ATAT = Average Turnaround Time

$$ATAT = \frac{Sum\ of\ all\ processes\ turnaround\ time}{number\ of\ processes} \quad 4.4$$

c. Response Time (RT): Amount of time it takes from when a request was submitted until the first response occurs (but not the time it takes to output the entire response). The formula to calculate response time is as shown below:

$$RT = Time\ of\ process\ fist\ response - Arrival\ time \quad 4.5$$

ART = Average Response Time

$$ART = \frac{Sum\ of\ all\ processes\ response\ time}{number\ of\ processes} \quad 4.6$$

4.6 Comparative Analysis

To compare the performance of the six algorithms, Table 4.8 shows the results where five processes are used for all the six algorithms, one hundred processes are used in Table 4.9 and One thousand processes are used in Table 4.10 respectively. The graphs of evaluation are shown in Figure 4.8, 4.9 and 4.10 respectively.

Table 4.8: Comparative table using 5 processes

Algorithms	AWT	ATAT	ART	NCS
FCFS	3.75	6.95	3.75	5
SJF	3.71	6.27	3.71	5
RR	6.55	9.75	2.25	10
IRR	4.35	7.55	6.63	7
DTSRRUBT	6.75	9.95	2.55	10
IDTSRRUBT	5.5	8.69	3.63	6

Table 4.9: Comparative Table using 100 Processes

Algorithms	AWT	ATAT	ART	NCS
FCFS	132.87	136.56	132.87	100
SJF	111.17	114.83	111.17	100
RR	211	214.68	80.83	322
IRR	205.88	209.57	108.24	182
DTSRRUBT	206.43	210.12	95.28	200
IDTSRRUBT	177.69	181.37	122.78	144

Table 4.10: Comparative Table using 1000 Process

Algorithms	AWT	ATAT	ART	NCS
FCFS	1332.29	1335.96	1332.29	1000
SJF	1097.3	1100.97	1097.3	1000
RR	2091.51	2095.18	791.95	2322
IRR	2029.7	2033.37	1066.82	1781
DTSRRUBT	2082.51	2086.19	933.78	2006
IDTSRRUBT	1748.56	1752.24	1221.44	1411

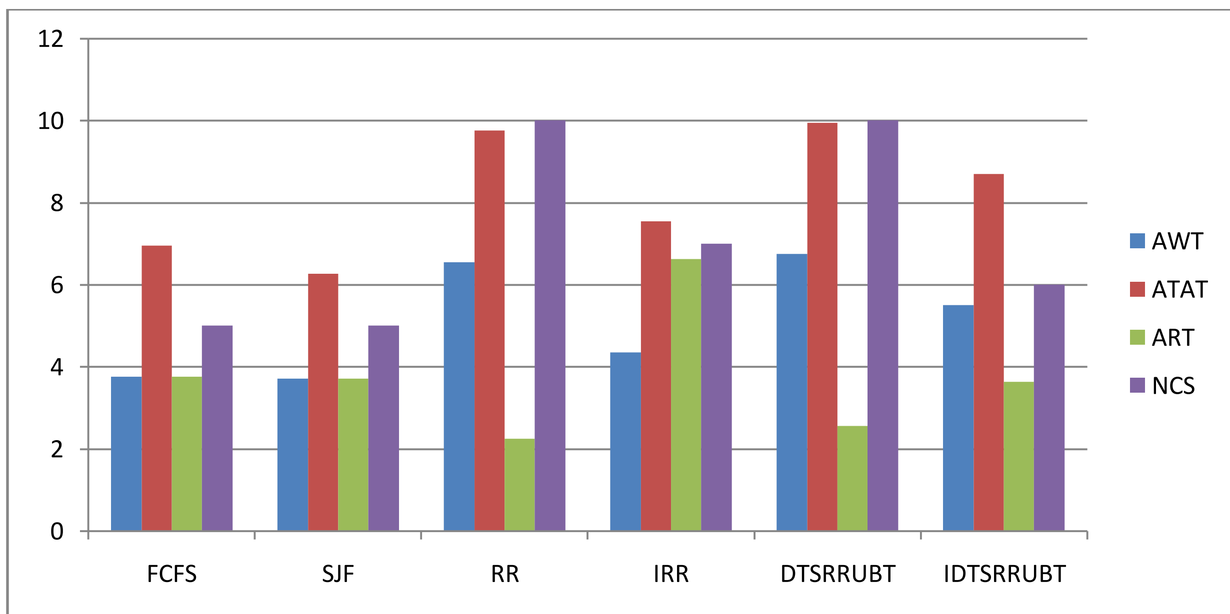

Figure 4.8: Graph of Evaluation Criteria using 5 Processes

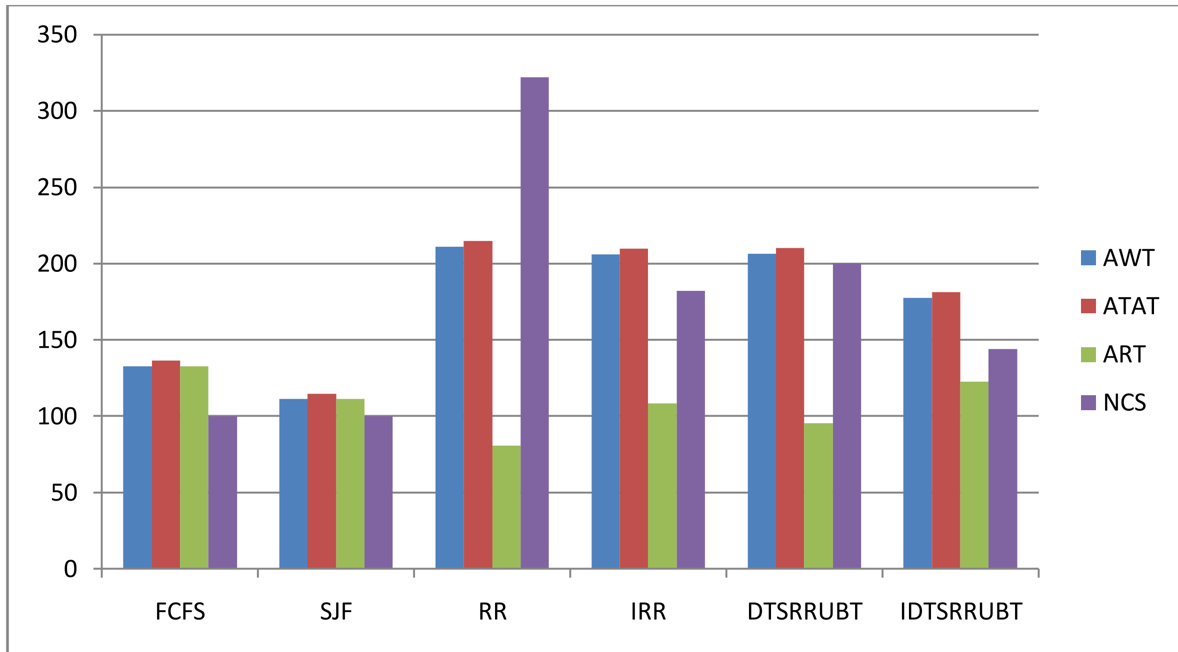

Figure 4.9: Graph of Evaluation Criteria using 100 Processes

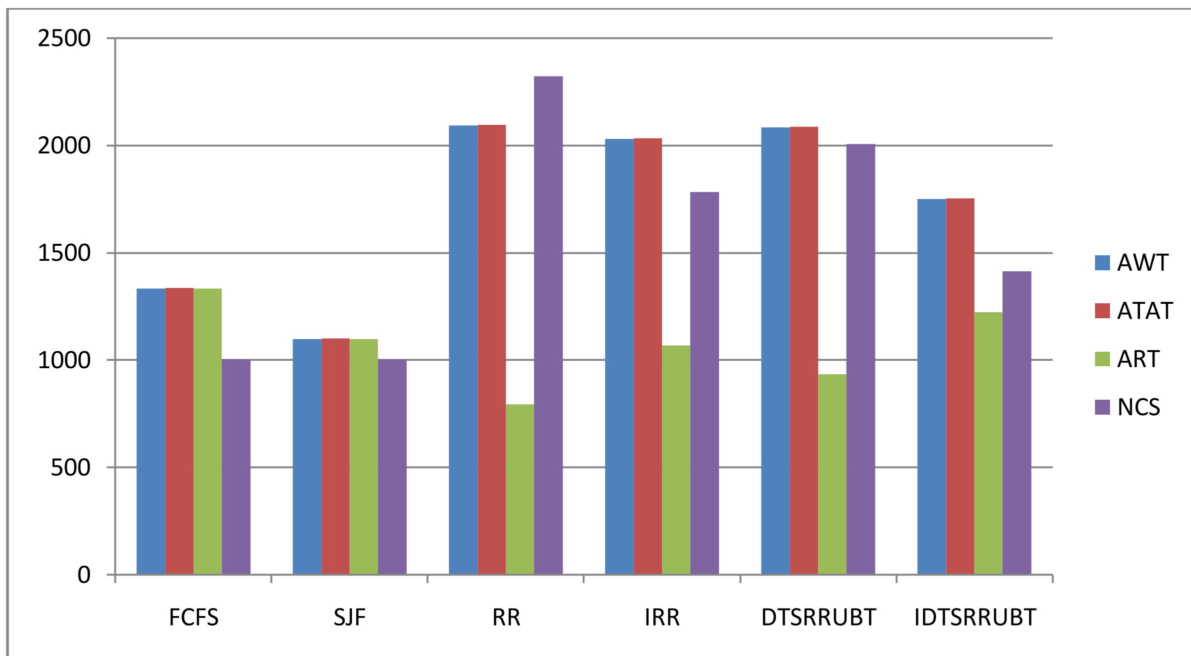

Figure 4.10: Graph of Evaluation Criteria using 1000 Processes

From the study of comparative tables and graph, it was discovered that the Improved Dynamic Time Slice Round Robin with unknown burst time (IDDTSRR) is better in term of average waiting time, average turnaround time and context switch. Apparently, if there are millions of processes the algorithm will still performed better.

It is desirable to minimize all the four performance criteria. The proposed algorithm when compared with the existing algorithms minimized the three out of the four criteria which makes it better than the existing Dynamic Time Slice Round Robin with Unknown Burst Time.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

This dissertationaimed to enhance dynamic time slice round robin with unknown burst time proposed by Anju, *et al.* (2016). The proposed algorithm solved the problem of assuming burst time, fixed quantum and random generation of arrival time. Processes arrive ready queue with their burst time and the details of each process. The first process is allocated to the CPU for a period of three 3 quantum time while subsequent quantum time is generated based on the average burst time of other processes. However, process is removed from ready queue when burst time is equal zero.

The performance of the proposed algorithm and 5 other scheduling algorithms (FCFS, SJF, RR, IRR and DTSRR) were experimented, analyzed and evaluated using simulation method. The results of the simulation areshowed in a tabular form for different scheduling algorithms and also presented graphically. The proposed algorithms performed better than all the other algorithms compared but DTSRR performs better in term of Response time.

5.2 Conclusion

In this dissertation, we have successfully presented an enhanced dynamic time slice round robin with unknown burst time scheduling algorithm in which from the experimental results, we found that it minimizes average waiting time, average turnaround time and number of context switches.

The aim of this dissertation has been achieved, which is the improvement of the performance of Anju, *et al.* (2016) algorithm. Also, the objectives of the dissertation for the attainment of the main aim were successfully achieved. The proposed algorithm was able to address the drawback of many algorithms in term of accurate determination of burst time for processes and can be used in a time sharing system.The algorithm was compared in term ofAverage Waiting Time, Number of Context Switch Turnaround Time and Response Time, but CPU utilization was not considered.

5.3 Recommendation

Future research should focus on investigating other method of determining burst time of processes to improve efficiency of scheduling techniques.

5.4 Contribution to Knowledge

This dissertation contributed to scheduling algorithm by introducing a method that generate burst time of process since many approach assumed burst time. The method also dynamically determines quantum time as the process arrived. The technique increases the performance of scheduling in term of average waiting time, average turnaround time and number of context switches.

REFERENCES

Abdulrazak, A., Abdullahi, S. E. and Sahalu, J. B. (2014). New Improved Round Robin (NIRR) CPU Scheduling Algorithm.*International Journal of Computer Applications* Volume 90 – No 4, pp. 27-33.

Achim, S. (2005).*Performance Evaluation of Slackness option for the Self-turning dynamic CPU scheduler*. Central Institute for Applied Mathematics.Julich, Germany: RSJ.

Adeli, A., Najaran, A., Sargolzaei, M. and Neshat, M. (2012) The New Method of Adaptive CPU Scheduling using Fonseca and Fleming's Genetic Algorithm.*Journal of Theoretical and Applied InformationTechnology*.37(1). Pp 1-16.

Aisha, A. S. (2016). *Improved Longest Job First CPU Scheduling Algorithm (ILJF)*.MSc Dissertation, Department of Mathematics, Ahmadu Bello University, Zaria, Nigeria.

Ajala, F. A., Fenwa, O. and Alade, M. O. (2015).A Comparative Analysis of Scheduling Algorithms.*International Journal of Research in Applied, Natural and Social Sciences*, 3(1),pp. 121-132.

Ajit, S, Priyanka, G and Sahil, B. (2010). An Optimized Round Robin Scheduling Algorithm for CPU Scheduling. *International Journal on Computer Science and Engineering, 2* (7), pp. 2382-2385.

Anju, M., Neenu, A., and Nandakumar, R (2016) Dynamic Time Slice Round Robin Scheduling Algorithm with Unknown Burst Time. *Indian Journal of Science and Technology*, Vol 9(8), pp. 1-6.

Ashiru, S., Saleh, A. and Junaidu, S. (2014a). **"**Dynamic Round Robin with Controlled Preemption (DRRCP)", IJCSI *International Journal of Computer Science Issues*, Vol. 11, Issue 3, No 1, pp. 5-27.

Ashiru, S., Saleh A. and Junaidu, S. (2014b). **"**Half Life Variable quantum Time Round Robin (HLVQTRR)",*IJCSIT International Journal of Computer Science and Information Technologies,* Vol. 5 (6), pp 7-23

Behera, H.S, Mohanty, R and Debashree, N. (2010). A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis. *International Journal of Computer Applications*, 5(5), pp. 10-15.

Dhal, S., Banerjee, P. and Banerjee, S. (2012) Comparative Performance Analysis of Average Max Round Robin Scheduling Algorithm (AMRR) using Dynamic Time Quantum with Round Robin Scheduling Algorithm using static Time Quantum. *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* (1)-3, pp 56-62.

Faiza T. B. (2016) *Design and Analysis of Improved Round Robin Algorithms with Arrival Time*. MSc Dissertation, Department of Mathematics, Ahmadu Bello University, Zaria, Nigeria.

Farooqui, M. Z. and Shoaib, M. (2014) *A Comparative Review of CPU Scheduling Algorithms* Mangalayatan University, Aligarh, India. Pp 20-28.

Ghani, R. F., Mustafa, B. A, Abdulmajid, M. and Mohammed, M. A. (2015) CPU Burst Processes Prioritization Using Priority Dynamic Quantum Time Algorithm: A Comparison with Varying Time Quantum and Round Robin Algorithms. International *Journal of Electrical & Computer Sciences* IJECS-IJENS (15)6.

Gupta, D., and Rajput, I. S. (2012) A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems. *International Journal of Innovations in Engineering and Technology (IJIET).*

Hamad, S. H., Mostafa, S., and Rida, S. Z. (2010) Finding Time Quantum of Round Robin CPU Scheduling Algorithm in General Computing Systems using Integer Programming. IJRRAS, 5(1) pp 65-70.

Ishwari, S. R and Deepa, G. (2012). A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems. *International Journal of Innovations in Engineering and Technology*, 1 (3), 1-11.

Jain, P., Bagish, A. D and Arora, H. (2013) An Improved CPU Scheduling Algorithm.*International Journal of Applied Information Systems* (IJAIS), (6) 6,pp 1-3.

Jerry, B. and John, S. C. (2009) *Discrete Event System Simulation* (5th Edition). Prentice Hall, Leemis.

Kadry, S., Kalakech, A. and Noon, A. (2011) A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average. *IJCSIInternational Journal of Computer Science* Issues, (8)3, pp 224-229.

Karlin, S. and Taylor, H. M. (1998).*An Introduction to Stochastic Modeling*, Academic Press.ISBN 0-12-684887-4.

Kendall, D. G. (1953). Stochastic *Processes* Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24 (3), pp. 338-354.

Manish, K. M., and Abdulkadir, K. (2012)An Improved Round Robin CPU Scheduling Algorithm.*Journal of Global Research in Computer Science*, 3 (6),pp 64-69.

Manish, K. M. and Faizur R. (2014) An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum.International Journal of Computer Science, Engineering and Applications (IJCSEA) (4)4.

Mittal, K. P. and Raman, (2014). An Efficient Dynamic Round Robin CPU Scheduling Algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering* (4) 5, pp 906-910.

Nirvikar and Kumar, N. (2013) Performance Improvement Using CPU Scheduling Algorithm-SRT. International *Journal of Emerging Trends & Technology in Computer Science (IJETTCS),* (2)2,pp. 110-113.

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

Ragha, L., Dhumal, R. A and Maktum, T. A. (2014) Dynamic Quantum based Genetic Round Robin Algorithm.*International Journal of Advanced Research in Computer and Communication Engineering* (3)3, pp 5905-5908.

Sahoo, S., Behera, H. S., Mohanty, R., Panda, J. and Thakur, P. (2011) Experimental Analysis of New Fair-Share Scheduling Algorithm with Weighted Time Slice for Real Time Systems. *Journal of Global Research in Computer Science*, Volume 2, No. 2, pp 15-36.

Sahu, G., Anmol, K. P., Brajendra, K. S., and Behera, H. S (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.

Sarma et al., (2006) "Wafer Level Reliability Application to Manufacturing of High Performance Microprocessor", *IEEE International Integrated Reliability Workshop*, Oct. 20, 1996, pp. 77-81.

Sendre, R. (2015). A Survey on Research Trends for Modeling CPU Scheduling Algorithm. *International Journal in IT and Engineering,* 3(7), pp. 1 -10.

Shaaban, (2010) CPU Performance Evaluation: Cycle Per Instruction (CPI). EECC550. *3rd and 4th Edition, Chapter 1 & 4.*

Sharma, H., Kumar, R. Y., Mishra K. A and Prakash N. (2010).An Improved Round Robin Scheduling Algorithm for CPU scheduling” *(IJCSE) International Journal on Computer Science and Engineering* (2)4, pp. 1064-1066.

Sharma, L. and Dhakad, V. K. (2012) Performance Analysis Of Round Robin Scheduling Using Adaptive Approach Based on Smart Time Slice And Comparison with SRR. *International Journal of Advances in Engineering & Technology*, 3(2), pp. 333-339.

Silberschatz, A., Galvin, P. B. and Gagne, G. (2013). Operating System Concepts. (9th, Ed.) John Wiley and Sons Inc, USA.

Singla, P. and Kanger, R. (2013) Performance Evaluation of CPU Scheduling Technique With an Efficient Genetic Algorithm. *International Journal of Computer Science And Technology IJCST* (4)3, pp. 82-88.

Sonagara, P. T. (2014) Round Robin CPU Scheduling Using Dynamic Time Quantum with Multiple Queue. *International Journal of Science and Research* (IJSR) Vol(1), pp. 2319-7064.

Sudhashree, Prasanma, M. L., Das, M., and Mohaty, R. (2011) Design and Performance Evaluation of A New Proposed Fittest Job First Dynamic Round Robin (FJFDRR) Scheduling Algorithm. *International Journal of Computer Information Systems*, Vol. 2, pp 23-27.

APPENDIX

```
Package com.web.sculpture.base.model;

importjavafx.beans.property.SimpleStringProperty;

public class DTOClockcyclecount {
	privateSimpleStringProperty id;
	privateSimpleStringPropertyinstructiontype;
	privateSimpleStringPropertyclockcyclecount;

	publicDTOClockcyclecount(String id, String instructiontype, String clockcyclecount) {
		this.id = new SimpleStringProperty(id);
		this.instructiontype = new SimpleStringProperty(instructiontype);
		this.clockcyclecount = new SimpleStringProperty(clockcyclecount);
	}

	public String getId() {
		returnid.get();
	}

	public String getInstructiontype() {
		returninstructiontype.get();
	}

	public String getClockcyclecount() {
		returnclockcyclecount.get();
	}

	@Override
	public String toString() {
		return "Process ID: " + getId() + "; Instruction Type: " + getInstructiontype() + ";
	Clock Cycle Count: "
				+ getClockcyclecount();
packagecom.web.sculpture.base.model;

importjavafx.beans.property.SimpleStringProperty;

public class DTOProcessdetailsrect {
	privateSimpleStringPropertyserialno;
	privateSimpleStringPropertyinstructiontype;
	privateSimpleStringPropertyinstructioncount;
	privateSimpleStringPropertyclockcyclecount;

	publicDTOProcessdetailsrect(String serialno, String instructiontype, String
	instructioncount,
```

```
                    String clockcyclecount) {

            this.serialno = new SimpleStringProperty(serialno);
            this.instructiontype = new SimpleStringProperty(instructiontype);
            this.instructioncount = new SimpleStringProperty(instructioncount);
            this.clockcyclecount = new SimpleStringProperty(clockcyclecount);
    }

    public String getSerialno() {
            returnserialno.get();
    }

    public String getInstructiontype() {
            returninstructiontype.get();
    }

    public String getInstructioncount() {
            returninstructioncount.get();
    }

    public String getClockcyclecount() {
            returnclockcyclecount.get();
    }

Package com.web.sculpture.base.function;

publicenumSimulationType {
    FCFS, SJF, RR, IRR, DTSRR, IDTSRR;
}
packagecom.web.sculpture.base.controller;

import java.net.URL;
importjava.util.ResourceBundle;

importcom.web.sculpture.base.data.DataUtils;
importcom.web.sculpture.base.function.AppUtil;
importcom.web.sculpture.base.function.SimulationType;
importcom.web.sculpture.base.model.DTOSimulation;
importcom.web.sculpture.base.sim.DTSim;
importcom.web.sculpture.base.sim.FCSim;
importcom.web.sculpture.base.sim.IDSim;
importcom.web.sculpture.base.sim.IRSim;
importcom.web.sculpture.base.sim.RRSim;
importcom.web.sculpture.base.sim.SJSim;
importcom.web.sculpture.base.view.PopupContener;
```

```
importjavafx.collections.FXCollections;
importjavafx.collections.ObservableList;
importjavafx.event.ActionEvent;
importjavafx.fxml.FXML;
importjavafx.fxml.Initializable;
importjavafx.scene.control.Button;
importjavafx.scene.control.ComboBox;
importjavafx.scene.control.Label;
importjavafx.scene.control.TableColumn;
importjavafx.scene.control.TableView;
importjavafx.scene.control.cell.PropertyValueFactory;
importjavafx.scene.input.MouseEvent;

@SuppressWarnings("rawtypes")
public class WinnrunsimulationController implements Initializable {

    privateObservableList<DTOSimulation>txSimulatorData                    =
    FXCollections.observableArrayList();

    @FXML
    privateComboBox<SimulationType>cbcSimulatortype;

    @FXML
    private Button btnRunSimulator;

    @FXML
    private Label lblsimulationdetails;

    @FXML
    privateTableView<DTOSimulation>tblSimulationDatarect;

    @FXML
    privateTableColumnclnProcessId;

    @FXML
    privateTableColumnclnArivaltime;

    @FXML
    privateTableColumnclnBurtstime;

    @FXML
    privateTableColumnclnFinishtime;

    @FXML
    privateTableColumnclnStarttime;
```

```
@FXML
privateTableColumnclnWaitingtime;

@FXML
privateTableColumnclnTurnarountime;

@FXML
privateTableColumnclnResponsetime;

@FXML
privateTableColumnclnContextSwitch;

@SuppressWarnings("unchecked")
@Override
public void initialize(URL location, ResourceBundle resources) {

        cbcSimulatortype.setItems(DataUtils.get().getComboData());
        lblsimulationdetails.setText(DataUtils.get().getAlgorithmsRuntype()        +        "
Simulation Details");

        clnProcessId.setCellValueFactory(new    PropertyValueFactory<DTOSimulation,
String>("processId"));
        clnArivaltime.setCellValueFactory(new   PropertyValueFactory<DTOSimulation,
String>("arrivaltime"));
        clnBurtstime.setCellValueFactory(new    PropertyValueFactory<DTOSimulation,
String>("bursttime"));
        clnStarttime.setCellValueFactory(new    PropertyValueFactory<DTOSimulation,
String>("starttime"));
        clnFinishtime.setCellValueFactory(new   PropertyValueFactory<DTOSimulation,
String>("finishedtime"));
        clnWaitingtime.setCellValueFactory(new
PropertyValueFactory<DTOSimulation, String>("waitingtime"));
        clnTurnarountime.setCellValueFactory(new
PropertyValueFactory<DTOSimulation, String>("turnaroundtime"));
        clnResponsetime.setCellValueFactory(new
PropertyValueFactory<DTOSimulation, String>("responsetime"));
        clnContextSwitch.setCellValueFactory(new
PropertyValueFactory<DTOSimulation, String>("contextswitch"));

        AppUtil.getIns().setSortable(clnProcessId,       clnArivaltime,       clnBurtstime,
clnStarttime, clnFinishtime,
                clnWaitingtime, clnTurnarountime, clnResponsetime);

        btnRunSimulator.addEventHandler(MouseEvent.MOUSE_CLICKED, evt -> {
                txSimulatorData.clear();
```

```
			SimulationTypetxSimulatortype =
	cbcSimulatortype.getSelectionModel().getSelectedItem();
			SHOWMessage.getInstance().getlblMessage().textProperty().unbind();
			if (txSimulatortype == null) {
				SHOWMessage.getInstance().showMessage("Select Simulator
	Type");
			} else {
				SHOWMessage.getInstance().showMessage("");
				lblsimulationdetails.setText(txSimulatortype + " Simulation
	Details");
				runSimulatn(txSimulatortype);
			}
		});
		runSimulatn(DataUtils.get().getAlgorithmsRuntype());
	}

	@FXML
	public void showSimulationGraph(ActionEvent event) {
		PopupContener.get().showPopup();
	}

	private void runSimulatn(SimulationTypestrData) {
		switch (strData) {
		case DTSRR:
			new Thread(new DTSim(this.tblSimulationDatarect)).start();
			break;
		case FCFS:
			new Thread(new FCSim(this.tblSimulationDatarect)).start();
			break;
		case IRR:
			new Thread(new IRSim(this.tblSimulationDatarect)).start();
			break;
		case IDTSRR:
			new Thread(new IDSim(this.tblSimulationDatarect)).start();
			break;
		case RR:
			new Thread(new RRSim(this.tblSimulationDatarect)).start();
			break;
		case SJF:
			new Thread(new SJSim(this.tblSimulationDatarect)).start();
			break;
		default:
			break;
		}
	}
}
```

```
public class IDSim extends Task<String> {

        private double avrWaintingTime = 0.0d;
        private double avrWTurnaroundTime = 0.0d;
        private double avrResponseTime = 0.0d;
        privateintcontextSwitch = 0;
        privateTableView<DTOSimulation>tblSimulationDatarect;
        privateXYChart.Series<String, Number>strWaitingtime = new XYChart.Series<>();
        privateXYChart.Series<String, Number>strTurnArrtime = new XYChart.Series<>();
        privateXYChart.Series<String, Number>strContextSwitch = new XYChart.Series<>();
        privateLinkedHashMap<String, Double>strBursttime = new LinkedHashMap<String,
        Double>();
        privateLinkedHashMap<String, Double>strBursttimeInit = new LinkedHashMap<String,
        Double>();
        privateLinkedHashMap<String, Double>strMainBursttime = new
        LinkedHashMap<String, Double>();
        privateLinkedHashMap<String, Double>strArivaltime = new LinkedHashMap<String,
        Double>();
        privateLinkedHashMap<String, Object>strProcdetails = new LinkedHashMap<String,
        Object>();
        privateObservableList<DTOSimulation>txSimulatorData =
        FXCollections.observableArrayList();

        privateLineChart<String, Number>strLineCharGraph;
        privateCategoryAxisstrXAxis = new CategoryAxis();
        privateNumberAxisstrYAxis = new NumberAxis();

        publicIDSim(TableView<DTOSimulation>tblSimulationDatarect) {
                this.tblSimulationDatarect = tblSimulationDatarect;
                strXAxis.setLabel("Process ID");
                strLineCharGraph = new LineChart<String, Number>(strXAxis, strYAxis);
                PopupContener.get().setCharGraph(strLineCharGraph);
                strLineCharGraph.setTitle("Improve Dynamic Time Slice Round Robin
        Algorithms");
        }

        @SuppressWarnings("unchecked")
        @Override
        protected String call() throws Exception {

                strWaitingtime.setName("Waiting Time");
                strTurnArrtime.setName("Turnarround Time");
                strContextSwitch.setName("Context Switch");

                AppUtil.getIns().addMapData(DataUtils.get().getBursttime(), strBursttime);
                AppUtil.getIns().addMapData(DataUtils.get().getBursttime(), strBursttimeInit);
```

```
        AppUtil.getIns().addMapData(DataUtils.get().getBursttime(), strMainBursttime);
        AppUtil.getIns().addMapData(DataUtils.get().getArivaltime(), strArivaltime);
        tblSimulationDatarect.setEditable(false);
        Iterator<String> iterator = strBursttimeInit.keySet().iterator();
        while (iterator.hasNext()) {
                String strKey = iterator.next();
                StringBuilder<ProcessBeans>jsonALrrdatarect = new
StringBuilder<ProcessBeans>();
                strProcdetails.put(strKey, jsonALrrdatarect);
        }
        Queue<String>strRequestQueue = new LinkedBlockingQueue<>();
        booleanstrLoop = true;
        doublestrQuantum = 3.0d;
        doublestrTime = 0;
        doublestrFishTime = 0;
        if (Double.valueOf(strBursttime.get("P1").toString()) > 3.0d) {
                strTime = 3.0d;
                strFishTime = 3.0d;
        } else {
                strTime = Double.valueOf(strBursttime.get("P1").toString());
                strFishTime = Double.valueOf(strBursttime.get("P1").toString());
        }

        StringBuilder<ProcessBeans>jsonArrdatarect = (StringBuilder<ProcessBeans>)
strProcdetails.get("P1");
        jsonArrdatarect.add(new ProcessBeans(0, strFishTime, 1));
        strProcdetails.put("P1", jsonArrdatarect);
        strBursttime.remove("P1");
        Iterator<Entry<String, Double>>strIterato = strBursttime.entrySet().iterator();
        while (strIterato.hasNext()) {
                Entry<String, Double> entry = strIterato.next();
                String strKey = entry.getKey();
                if (strQuantum>= strArivaltime.get(strKey)) {
                        strRequestQueue.add(strKey);
                        strIterato.remove();
                } else {
                        break;
                }
        }
        doublestrFirstJobArri = strBursttimeInit.get("P1") - strQuantum;
        if (strFirstJobArri> 0) {
                strBursttimeInit.put("P1", strFirstJobArri);
                strRequestQueue.add("P1");
        }
        while (strLoop) {
                doublestrQuant = 3.0d;
```

```
		if (strBursttime.size() > 0) {
			for (String strString : strRequestQueue) {
				strQuant += strMainBursttime.get(strString);
			}
			strQuantum = new Double(
					AppUtil.getIns().roundUPData(strQuant                /
Double.valueOf(strRequestQueue.size())));
		}
		String strProID = strRequestQueue.remove();
		if (strBursttimeInit.get(strProID) >= strQuantum) {
			doublestrJobArrivalLoop     =     strBursttimeInit.get(strProID)     -
strQuantum;
			strBursttimeInit.put(strProID, strJobArrivalLoop);
			StringBuilder<ProcessBeans>jsonArrdatarectLoop                =
(StringBuilder<ProcessBeans>) strProcdetails
					.get(strProID);
			jsonArrdatarectLoop.add(new  ProcessBeans(strTime,  (strTime  +
strQuantum), 1));
			strProcdetails.put(strProID, jsonArrdatarectLoop);
			strTime += strQuantum;
			if (strJobArrivalLoop> 0) {
				strRequestQueue.add(strProID);
			}
		} else {
			StringBuilder<ProcessBeans>jsonSectArrdatarectLoop              =
(StringBuilder<ProcessBeans>) strProcdetails
					.get(strProID);
			jsonSectArrdatarectLoop.add(new ProcessBeans(strTime, (strTime
+ strBursttimeInit.get(strProID)), 1));
			strProcdetails.put(strProID, jsonSectArrdatarectLoop);
			strTime += strBursttimeInit.get(strProID);
		}
		Iterator<Entry<String,                    Double>>strIterar              =
strBursttime.entrySet().iterator();
		while (strIterar.hasNext()) {
			Entry<String, Double> entry = strIterar.next();
			String strDKey = entry.getKey();
			if (strTime>= strArivaltime.get(strDKey)) {
				strRequestQueue.add(strDKey);
				strIterar.remove();
			}
		}
		if (strRequestQueue.size() == 0) {
			strLoop = false;
		}
	}
```

```
        Iterator<String>strIteLen = strProcdetails.keySet().iterator();
        while (strIteLen.hasNext()) {
                intcontenPswitch = 0;
                String strkey = strIteLen.next();
                Thread.sleep(10);
                StringBuilder<ProcessBeans>jsonSectArrdatarectLoop                    =
(StringBuilder<ProcessBeans>) strProcdetails
                                .get(strkey);
                finalintllen = jsonSectArrdatarectLoop.getSize();
                doublestrWaitingTime = 0;
                for (intiI = 0; llen>iI; iI++) {
                        ProcessBeansstrBeans = jsonSectArrdatarectLoop.get(iI);
                        contenPswitch += Integer.valueOf(strBeans.getContextswitch());
                        if (iI == 0) {
                                strWaitingTime                                        +=
Double.valueOf(strBeans.getStrStarttime());
                        } else {
                                ProcessBeansstrSectBeans                              =
jsonSectArrdatarectLoop.get(iI - 1);
                                doublestrWait                                         =
Double.valueOf(strBeans.getStrStarttime())
                                                                        -
Double.valueOf(strSectBeans.getStrFinishtime());
                                strWaitingTime                                        +=
Double.valueOf(AppUtil.getIns().roundUPData(strWait));
                        }
                }
                doublestrMwait = Double.valueOf(
                                AppUtil.getIns().roundUPData(strWaitingTime           -
Double.valueOf(strArivaltime.get(strkey))));
                doubletxfinishedtime                                                  =
Double.valueOf(AppUtil.getIns().roundUPData(Double

        .valueOf(jsonSectArrdatarectLoop.get(jsonSectArrdatarectLoop.getSize()        -
1).getStrFinishtime())));
                doubletxStarttime = Double

        .valueOf(AppUtil.getIns().roundUPData(jsonSectArrdatarectLoop.get(0).getStrSt
arttime()));
                doubleturnArtime = Double.valueOf(AppUtil.getIns()
                                .roundUPData(txfinishedtime                           -
Double.valueOf(strArivaltime.get(strkey).toString())));
                doubleresponseTime = Double.valueOf(AppUtil.getIns()
                                .roundUPData(txStarttime                              -
Double.valueOf(strArivaltime.get(strkey).toString())));
```

```
                contextSwitch += contenPswitch;
                finalintstrConswitch = contenPswitch;
                Platform.runLater(new Runnable() {
                        @Override
                        public void run() {
                                strWaitingtime.getData().add(new    XYChart.Data<String,
Number>(strkey, strMwait));
                                strTurnArrtime.getData().add(new    XYChart.Data<String,
Number>(strkey, turnArtime));
                                strContextSwitch.getData().add(new XYChart.Data<String,
Number>(strkey, strConswitch));
                        }
                });
                SimBeanssimBeans    =    new    SimBeans(turnArtime,    strMwait,
responseTime, contextSwitch);
                SimData.getData().getStrIDTSRR().add(simBeans);
                DTOSimulationtxSimdatarect    =    new    DTOSimulation(strkey,
strArivaltime.get(strkey),
                                strMainBursttime.get(strkey),  txStarttime,  txfinishedtime,
strMwait, turnArtime, responseTime,
                                contenPswitch);
                txSimulatorData.add(txSimdatarect);

                avrWaintingTime += strMwait;
                avrWTurnaroundTime += turnArtime;
                avrResponseTime += responseTime;
                updateValue(txSimdatarect.toString());
        }
        Platform.runLater(new Runnable() {
                @Override
                public void run() {
                        strLineCharGraph.getData().addAll(strWaitingtime,
strTurnArrtime, strContextSwitch);
                }
        });
        return "Done";
}

@Override
protected void updateValue(String txSimData) {
        Platform.runLater(new Runnable() {
                @Override
                public void run() {
                        finalintlen = txSimulatorData.size() - 1;
                        tblSimulationDatarect.requestFocus();
                        tblSimulationDatarect.setItems(txSimulatorData);
```

```
                        tblSimulationDatarect.scrollTo(len);
                    }
                });
            }

            @Override
            protected void succeeded() {
                DTOSimulationtxSimdatarectAvr = new DTOSimulation("Average", "", "", "",
            "",
                        AppUtil.getIns().roundUPData(avrWaintingTime                /
            Double.valueOf(strMainBursttime.size())),
                        AppUtil.getIns().roundUPData(avrWTurnaroundTime             /
            Double.valueOf(strMainBursttime.size())),
                        AppUtil.getIns().roundUPData(avrResponseTime                /
            Double.valueOf(strMainBursttime.size())),
                        contextSwitch);
                txSimulatorData.add(txSimdatarectAvr);
                tblSimulationDatarect.setItems(txSimulatorData);
            }
        }
```