A NEW IMPROVED ROUND ROBIN (NIRR) CPU SCHEDULING ALGORITHM

By

ABDULRAZAQ ABDULRAHIM

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

SEPTEMBER, 2014.

A NEW IMPROVED ROUND ROBIN (NIRR) CPU SCHEDULING ALGORITHM

By

ABDULRAZAQ ABDULRAHIM, B.Sc Computer Science (ABU) 2008

MSc/SCI/03673/09-10

A THESIS SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES,

AHMADU BELLO UNIVERSITY, ZARIA

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

DEPARTMENT OF MATHEMATICS,

FACULTY OF SCIENCE

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

SEPTEMBER, 2014

DECLARATION

I declare that the work in this project report entitled A New Improved Round Robin (NIRR) CPU Scheduling Algorithm has been performed by me in the Department of Mathematics under the supervision of Dr. S.E Abdullahi and Professor S.B Junaidu. The information derived from the literature has been duly acknowledged in the text and list of references provided. No part of this project report was previously presented for another degree or diploma at any university.

ABDDULRAHIM ABDULRAZAQ	____________________	____________________
Name of student	Signature	Date

CERTIFICATION

This thesis entitled A NEW IMPROVED ROUND ROBIN (NIRR) CPU SCHEDULING ALGORITHM by Abdulrazaq, ABDULRAHIM meets the regulation governing the award of the degree of M.Sc Computer Science of Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

(Signature) ____________________

____________________ Date ____________________

Chairman, Supervisory Committee

(Signature) ____________________

____________________ Date ____________________

Member, Supervisory Committee

(Signature) ____________________

____________________ Date ____________________

Member, Supervisory Committee

(Signature) ____________________

____________________ Date ____________________

Head of Department

(Signature) ____________________

____________________ Date ____________________

Dean, School of Postgraduate Studies

DEDICATION

I dedicate this work to Almighty Allah (S.W.A), for seeing me through in all things. And also to my beloved parents in the name of Alhaji Bashiru Abdulrahim and Hajia Wosilat Bashiru Abdulrahim for not only their financial support but for the time and resources they spent in training and bringing me up to be a person of value, importance to the society and life in general.

ACKNOWLEDGEMENT

In the name of Allah, the Most Gracious, the Most Merciful. All praise and thanks are Allah's for giving me the health and intellect to see the actualization of the work. My humble salutation to my Noble prophet Muhammad (S.A.W) and members of his household

First and foremost, I want to register my appreciation with thanks and due respect to my humble supervisors Dr. S.E Abdullahi (Major) and Professor S.B Junaidu (Minor) for supervising this work. In fact you did more than supervisors will ordinarily do. For their positive comments, suggestions and general concern which finally brought this work to this stage, I am immensely thankful. I also acknowledge the effort of the entire lecturers of the Department of Mathematics, Ahmadu Bello University Zaria, for their academic and intellectual support towards the success of this thesis.

A special acknowledgment goes to my family, Bro. Jimoh, Amina, Isiaka, Lawal, Saeed, Suleiman and Ameerat Abdulrahim and also, to my Uncle Engr. Abdulwasiu for their financial and moral support for easy running of my entire study programme. I cannot forget mentioning some of my colleagues who contributed one way or the other towards the completion of this thesis, people like Salihu Idi Dishing, A. Salisu, A. Abdul Jafar, Baba Muhammad Ahmad Mustapha, A. Baba Abu and my entire class members.

My sincere gratitude to Dr. K.M Lawal, Dr. T.O Ahmad and Dr. A.A Abdelmalik all of Department of Physics, Ahmadu Bello University, Zaria and Dr. Umar Faruq Isa of Department of Geography, Ahmadu Bello University, Zaria.

A special acknowledgment also goes to my big Brother; Raimi Jimoh for your academic, moral and financial support towards the successful completion of this thesis.

Finally, I thank you all. May Almighty Allah (S.W.A) be with you all in all your endeavors. Ameen

ABSTRACT

The Round Robin CPU scheduling algorithm is a fair scheduling algorithm that gives equal time quantum to all processes. The choice of the time quantum is critical as it affects the algorithm's performance. This thesis proposes a new algorithm that further improved on the Improved Round Robin (IRR) CPU scheduling algorithm by Manish and AbdulKadir. This proposed algorithm was implemented and benchmarked against First-Come-First-Serve (FCFS), Shortest-Job-First (SJF), Round Robin (RR), Improved Round Robin (IRR) and Longest Job First with Combinational Burst Time (LJF+CBT) using three different statistical distributions (namely Normal, Uniform and Exponential distributions) to generate the burst time of the processes. It was observed that the proposed algorithm (i.e. A New Improved Round Robin (NIRR)) compared with the other Round Robin scheduling algorithms, produces best Average Waiting Time (AWT), Average Turnaround Time (ATAT) and Number of Context Switches (NCS) in all categories of the statistical distributions used. Based on these results, the proposed algorithm outperformed other scheduling algorithms for systems that adopt RR CPU scheduling.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ART: Average Response Time

ATAT: Average Turn-Around Time

AWT: Average Waiting Time

BW: Burst Time Priority Weight

CBT: Combinational Burst Time

CPU: Central Processing Unit

CS: Context Switch

CWA: Combined Weighted Average

DTQSJF: Dynamic Time Quantum in Shortest Job First Scheduling Algorithm

FCFS: First Come First Served

FIFO: First In First Out

FIS: Fuzzy Interface System

FJFDRR: Fittest Job First Dynamic Round Robin

HRR: Highest Response Ratio

HRRN: Highest Response Ratio Next

I/O: Input Output

ISRBRR: Improving Shortest Remaining Burst Round Robin

IRR: Improved Round Robin

LJF: Longest Job First

LJF+CBT: Longest Job First + Combinational Burst Time

NIRR: New Improved Round Robin

NCS: Number of Context Switches

OS: Operating System

PBSRR: Priority Based Static Round Robin

PCB: Process Control Block

PS: Processor Sharing

RR: Round Robin

RRHRRN: Round Robin with Highest Response Ratio Next

RT: Response Time

SJF: Shortest Job First

SJN: Shortest Job Next

SRBRR: Shortest Remaining Burst Round Robin

STS: Smart Time Slice

TAT: Turn-Around Time

TQ: Time Quantum

UP: User Priority

UW: User Priority Weight

WT: Waiting Time

CHAPTER ONE

INTRODUCTION

This chapter discusses the introductory part of this thesis which includes the background of the study, research motivations and goals which states the research questions for which the thesis should provide answers to, the research aim and objectives, the methodology that was used in answering those questions stated and finally the summary of the thesis contribution to knowledge.

1.1 Background of the Study

An operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the system hardware. Multiprogramming is one of the most important aspects of operating systems amongst multitasking, multithreading etc. A single user cannot, in general keep either the CPU or the input/output (I/O) devices busy at all times. Multiprogramming increases CPU utilization by organizing processes (code and data), so that the CPU always has one to execute (Silberschatz *et al*, 2006; William, 2012).

The idea is as follows: the operating systems keep several processes in memory simultaneously. The set of processes can be subsets of the processes in the process pool, which contains all processes that can enter the system. Since the number of processes that can be kept simultaneously in memory is usually smaller than the number of processes that can be kept in the process pools, the operating systems pick and begin to execute one of the processes in the memory. Eventually, the process may have to wait for some tasks, such as an I/O operation, to complete (Silberschatz *et al*, 2006; William, 2012).

In a non-multiprogrammed (single processing) systems, while the process is waiting for the tasks such as an I/O operation, any other task must wait until the CPU is free and can be rescheduled. In contrast, the multiple programming systems allow multiple programs to be loaded into memory and executed concurrently (Silberschatz *et al*, 2006).

Multiprogramming requires several processes to be kept simultaneously in memory. Since in general, the main memory is too small to accommodate all processes, the processes are kept initially on the disk in the process pool. This pool consists of all processes residing on the disk, awaiting allocation of main memory. If several processes are ready to be brought into the memory, and if there is no enough room for all of them, then the operating system must choose among them. Making this decision is called process scheduling. When the operating system selects a process from the process pool, it loads that process into memory for execution. If several processes are ready to run at the same time, the operating system must choose amongst them. Making this decision is called CPU scheduling (Silberschatz *et al*, 2006).

CPU scheduling is the basis of multiprogramming systems. It refers to a set of policies and mechanisms to control the order of work to be performed by a computer system. Scheduling is the method by which threads, processes or data flows are given access to system resources. The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously) (Ishwari and Deepa, 2012). The basic idea is to keep the CPU busy as much as possible by executing a user process until it must wait for an event, and then switch to another process. In multiprogramming systems, when there are more than one runnable process (i.e., ready), the operating system must decide which one to activate. The decision is made by the part of the operating system called the scheduler, using a scheduling algorithm (Suri and Sumit, 2012).

1.2 Research Motivation and Goals

CPU scheduling is the basis of multiprogramming; it switches the CPU among processes so frequently. Its aim is to assign processes to be executed by the CPU. It affects the performance of the system, because it determines which process will wait and which will progress (Silberschatz *et al*, 2006). There exist a number of CPU scheduling algorithms like First-Come-First-Serve, Shortest-Job-Scheduling, Round Robin scheduling and priority Scheduling, but due to a number of disadvantages these scheduling algorithms have, they are rarely used except Round Robin scheduling in timesharing and real time operating systems; and also, it is considered as the most widely used scheduling algorithm in timesharing and real time operating systems (Ajit *et al.*, 2010; Behera *et al.*, 2012; Ishwari and Deepa, 2012; Saeidi and Hakimeh, 2012; Soraj and Roy, 2012).

The performance of Round Robin Scheduling is sensitive to time quantum selection, because if time quantum is very large then Round Robin scheduling is same as the FCFS scheduling. If the time quantum is extremely too small then Round Robin scheduling is same as Processor Sharing algorithm and number of context switches is very high (Saeidi and Hakimeh, 2012; Soraj and Roy, 2012). Each value will lead to a specific performance and will affect the algorithm's efficiency by affecting the processes waiting time, turnaround time, response time, throughput, CPU utilization and number of context switch.

Manish and AbdulKadir (2012); developed an algorithm that describes improvement in Round Robin scheduling algorithm. The algorithm is good because; it drastically minimizes waiting time and turnaround time with respect to the simple Round Robin scheduling algorithm.

Consequently, the goal of this thesis work is to further modify the Improved Round Robin CPU scheduling algorithm by Manish and AbdulKadir (2012), in order to increase the performance of the Round Robin scheduling algorithm by affecting the average waiting time, average turnaround time and number of context switch positively.

1.3 Research Aim and Objectives

The aim of this thesis work is to develop an algorithm that will make improvements to the RR CPU scheduling algorithm. This will be done by modifying the Improved Round Robin scheduling algorithm by Manish and AbdulKadir (2012) and then simulate this modified algorithm and compare it with the FCFS, SJF, RR, the Improved Round Robin (IRR) scheduling algorithm by Manish and AbdulKadir (2012) and Longest Job First with Combinational Burst Time (CBT+LJF) scheduling algorithm by Abdullahi and Junaidu (2013). The following are the objectives of this research work:

1. Modification of the Improved Round Robin scheduling algorithm by Manish and AbdulKadir (2012).
2. Dynamic determination of time quantum from the available burst times of the processes in the ready queue.
3. Application of the determined time quantum in the Improved Round Robin scheduling algorithm (IRR)
4. Implementation of the modified Improved Round Robin scheduling algorithm.

5. Evaluation of the six implementations (i.e. FCFS, SJF, RR, IRR, LJF+CBT and the modified algorithm).

1.4 Research Methodology

The following are the steps that were adopted for this research work:

1. Intensive study of CPU scheduling algorithms
2. Review of literatures on Round Robin scheduling algorithm
3. Development of an algorithm to modify the IRR scheduling algorithm
4. Simulation of the modified CPU scheduling algorithm, FCFS, SJF, RR, IRR and LJF+CBT using Java programming
5. Random generation of data used in the simulation: the burst time of the processes were generated using either normal, uniform or exponential distribution and the arrival time was generated using Poisson distribution
6. The six algorithms (i.e. FCFS, SJF, RR, IRR and LJF+CBT and the modified algorithm) were compared based on Average Waiting time, Average Turn Around Time, Average Response Time and Number of Context Switch for the different category of burst times generated

1.5 Contribution to Knowledge

The thesis contributed to the:

1. Introduction of a new queue (known as ARRIVE QUEUE) to the process state diagram
2. Improvement of the RR scheduling algorithm by dynamically determining time quantum which when applied to the IRR scheduling algorithm in SJF manner, increased the performance of the RR scheduling algorithm by affecting the average waiting time, average turnaround time and number of context switches positively.

CHAPTER TWO

LITERATURE REVIEW

This chapter discusses the concepts of CPU scheduling in operating systems, which comprises of CPU/IO burst cycle, the process state, scheduling criteria and scheduling algorithms. It also discusses the concept of time quantum provision as the major drawback of Round Robin scheduling algorithm and surveys the work done by various researchers in the field. The surveys were carried out into two different categories: static provision of time quantum in the Round Robin scheduling algorithm and dynamic determination of time quantum in the Round Robin scheduling algorithm. It further discusses the Longest Job First with Combinational Burst Time CPU scheduling algorithm and finally discusses concept of queuing theory.

2.1 CPU Scheduling

The operating system is the most important program needed for starting up and using the hardware. The hardware has different managing tasks each of which is performed by one of the management units of the operating system. The process management, as one of these management units, allocates processes to the processor (Silberschatz *et al,* 2006).

A process is a program in execution. The process needs certain resources-including CPU time, memory, files, and I/O devices-to accomplish its task. These resources are either allocated to the process when it is created or allocated to it while it is running. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently that user can interact with each program while it is running. To meet these objectives, the process management uses several CPU scheduling algorithms to select an available process (possibly from a set of several available processes) for program execution on the CPU (Mohammed, 2011; Saeidi and Hakimeh, 2012).

2.1.1 CPU-I/O burst cycle

In the computer system, all processes consist of a number of alternating two burst cycles (the CPU burst cycle and the Input & Output (I/O) burst cycle) (Silberschatz *et al*, 2006). Normally, a process will run for a while (the CPU burst), perform some I/O (the I/O burst), then run for a while more (the next CPU burst), again perform some I/O (the I/O burst). These cycles continues until the execution of the process is completed (as shown in Figure 2.1). An I/O Bound process is a process that performs lots of I/O operations such as reading from and writing to disks. Each I/O operation is followed by a short CPU burst to process the I/O, and then more I/O happens. A CPU bound process is a process that performs lots of computation and do little I/O. A typical system has a few long CPU bursts. One of the things a scheduler will typically do is to switch the CPU to another process when one process does I/O operations. This is because the I/O usually takes a long time, and the CPU is not supposed to be idle while the process is waiting for the I/O to finish.

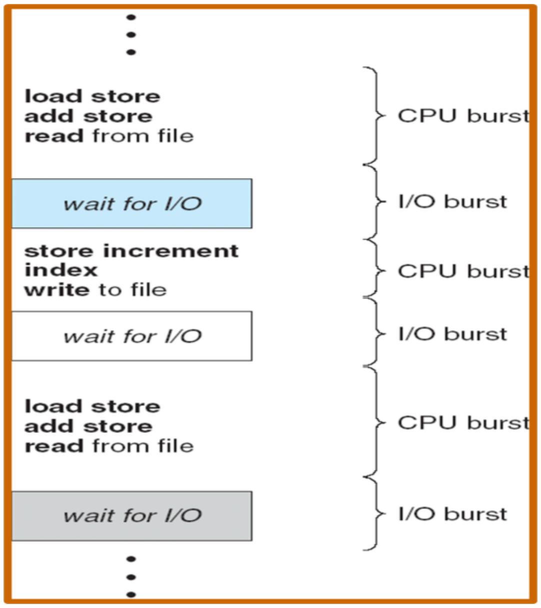


Figure 2.1: Diagram Illustrating Alternating Sequence of CPU and I/O Burst
(http://www.scs.stanford.edu/07au-cs140/notes/l5.pdf)

2.1.2 The 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. Each process may be in one of the following states (as shown in Figure 2.2) (Silberschatz *et al*, 2006):

1. New: the process is being created.
2. Running: instructions are being executed
3. Waiting: the process is waiting for some events to occur (such as an I/O completion or reception of signal)
4. Ready: the process is waiting to be assigned to the processor
5. Terminated: the process has finished execution

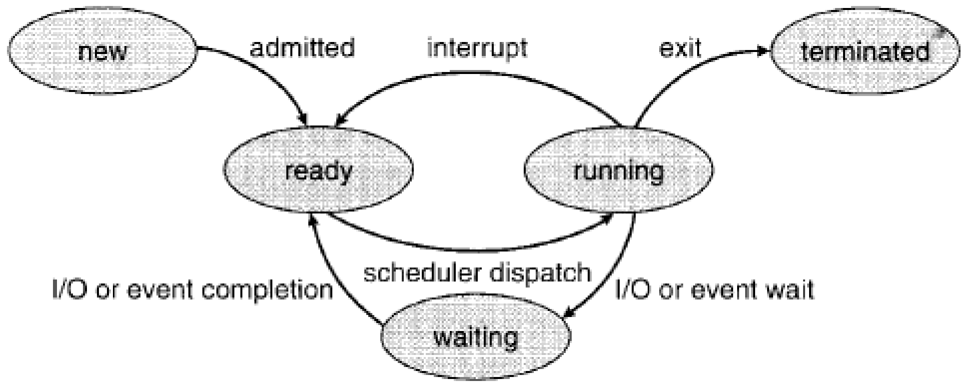


Figure 2.2: A Diagram Illustrating Process State (Frédéric, 2007)

The CPU chooses (schedules) which process to run when any of the following occurs:

1. When process switches from running to waiting. This could be as a result of I/O request, waiting for child to terminate, or waiting for synchronization operation to complete.
2. When process switches from running to ready. This could be as a result of completion of interrupt handler. If scheduler switches processes in this case, it has preempted the running process.
3. When process switches from waiting to ready state (on completion of I/O).
4. When a process terminates.

For situations 1 and 4 there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution. There is a choice, however, for situations 2 and 3 to either continue running the current process, or select a different one.

If scheduling takes place only under conditions 1 and 4, the system is said to be non-preemptive, or cooperative. Under these conditions, once a process starts running it keeps running, until it either voluntarily blocks or until it finishes. Otherwise the system is said to be preemptive (Silberschatz *et al*, 2006).

2.1.3 CPU scheduling algorithms

CPU Scheduling is the act of selecting the next process for the CPU to service, once the current process leaves the CPU. Some basic CPU scheduling algorithms are listed in the following subsections:

1. **First-Come First-Serve (FCFS)**

It is by far the simplest CPU scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of this algorithm is easily managed with a First-In-First-Out (FIFO) queue. When a process enters the ready queue, it is inserted onto the tail (rear) of the ready queue and when the CPU is free, the process to be executed next is removed from the head (front) of the ready queue. The processes are allocated to the CPU on the basis of their arrival to the ready queue (Abdulrazaq *et al.*, 2014). It has the following advantages:

1. It is simple and has low overhead (Insup, 2007)
2. The code for FCFS scheduling is simple to write and understand (Rashid, 2014)

It has the following disadvantages:

1. A long CPU-bound job may dominate the CPU and may force shorter jobs to wait prolonged periods (Silberschatz *et al,* 2006; Rahul and Umesh, 2012)
2. Convoy effect occurs. Even very small process should wait for its turn to come to utilize the CPU. Short process behind long process results in lower CPU utilization (Silberschatz *et al,* 2006; Rahul and Umesh, 2012).
3. Minimal average CPU utilization or average throughput (Oyetunji and Oluleye, 2009; Abdulrazaq *et al.*, 2014).

2. **Shortest-Job-First (SJF)**

This algorithm associates with each process the length of the next process's CPU burst time. When the CPU is available, it is assigned the process that has the least next CPU burst. If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

The SJF can be either preemptive or non-preemptive. The choice arises when a new process arrives in the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than the time remaining in the process whose burst is currently on the CPU. A preemptive SJF algorithm will preempt the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst (Oyetunji and Oluleye, 2009; Silberschatz *et al*, 2006). Preemptive SJF is sometimes referred to as shortest remaining time first scheduling (Silberschatz *et al*, 2006; William, 2012). It has the following advantages:

1. Provably optimal, in that it gives the minimum average waiting time and minimum average turnaround time for a given set of processes (Oyetunji and Oluleye, 2009; Rahul and Umesh, 2012; Insup, 2007).

It has the following disadvantages:

1. It is difficult to know the length of the next CPU burst (Silberschatz *et al*, 2006)
2. It cannot be implemented because of the difficulty in determining the shortest remaining burst time next (Insup, 2007)
3. Long running processes may starve, because the CPU has a steady supply of short processes (Suri and Sumit, 2012; Silberschatz *et al*, 2006; Rahul and Umesh, 2012; Insup, 2007).

3. **Priority Scheduling (PS)**

Priority scheduling is a more general case of SJF. In which each process is assigned a priority and the process with the highest priority gets CPU allocated to it first. Equal-priority processes are scheduled in FCFS order (Abdulrazaq *et al.*, 2014; Oyetunji and Oluleye, 2009). (SJF uses the inverse of the next expected burst time as its priority-The smaller the expected burst, the higher the priority). Note that in practice, priorities are implemented using integers within a fixed range, but there is no agreed-upon convention as to whether "high" priorities use large numbers or small numbers. Priorities can be defined either internally or externally. Internal defined priorities are assigned by the operating system by using measurable quantity or quantities to compute the priority of the process. Such quantities include average burst time, ratio of CPU to I/O activity, system resource use, and other factors available to the kernel. External priorities are set by criteria outside the operating system; such criteria include importance of the process, fees being paid for computer use, the department sponsoring the work, politics, etc (Silberschatz *et al*, 2006).

Priority scheduling can be either preemptive or non-preemptive. When a process arrives in the ready queue, its priority is compared with the priority of the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority currently running process. A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue (Silberschatz *et al*, 2006).

It has the following advantages:

1 Good response for the highest priority processes (Abdulrazaq *et al.*, 2014).

It has the following disadvantages:

1. It suffer from a major problem known as indefinite blocking, or starvation, in which a low-priority task can wait forever because there are always some other processes around that have higher priority. (Abdulrazaq *et al.*, 2014; www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html)

4 Round Robin Scheduling (RR)

Round Robin scheduling is designed for time-sharing systems (Mohammed, 2011). It is similar to FCFS scheduling, but preemption is added to the switch between processes. A small time unit called the time quantum or time slice is defined. The ready queue is maintained as a circular queue. The CPU scheduler goes round the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. To implement the Round Robin scheduling, we keep the ready queue as a First-In-First-Out (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 1 time quantum, and dispatches the process (Abdulrazaq *et al.*, 2014; Silberschatz *et al*, 2006).

One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running burst is longer than 1 time quantum, the timer goes off, and will cause interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process at the head of the ready queue. In the Round Robin scheduling algorithm, no process is allocated the CPU more than 1 time quantum in a row (unless it is the only runnable process). If a process's CPU burst exceeds 1 time quantum, that process will be preempted and is put in the ready queue. The Round Robin scheduling algorithm is thus preemptive (Abdulrazaq *et al.*, 2014; Silberschatz *et al*, 2006).

It has the following advantages (www.go4expert.com/articles/types-of-scheduling-t22307):

1. Round-Robin is effective in a general-purpose, times-sharing system or transaction-processing system.
2. Fair treatment for all the processes.
3. Overhead on processor is low.
4. Good response time for short processes.

It has the following disadvantages (www.go4expert.com/articles/types-of-scheduling-t22307):

1. Care must be taken in choosing quantum value.
2. Throughput is low if time quantum is too small

2.1.4 Scheduling criteria

The various CPU scheduling algorithms have different properties as mentioned in section 2.1.3, and the choice of a particular algorithm may favor one class of processes over another. For selection of an algorithm for a particular situation, the properties of various algorithms must be considered. Many criteria have been suggested for comparing CPU scheduling algorithms. These criteria are used for comparison and to make a substantial difference in which algorithm is judged to be the best. The criteria include the following (Ajit *et al*; 2010; Ramakrishna and Pattabhi, 2013):

1. Context Switch: A context switch is a 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 systems is to optimize only these switches, the goal is to minimize it.

2. Throughput: Throughput is defined as number of processes completed per unit time. Throughput is low in Round Robin scheduling implementation. Context switching and throughput are inversely proportional to each other.
3. CPU Utilization: This is a measure of how much busy the CPU is. Usually, the goal is to maximize the CPU utilization.
4. Turnaround Time: Turnaround time refers to the total time which is spent to complete the process and is how long it takes the CPU 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.
5. 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.
6. 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. So the response time should be low for best scheduling.

So, it can be concluded that a good scheduling algorithm for real time and time sharing system must possess the following characteristics (Ajit *et al*; 2010; Ramakrishna and Pattabhi, 2013):

1. Minimum context switches.
2. Maximum CPU utilization.
3. Maximum throughput.
4. Minimum turnaround time.
5. Minimum waiting time.
6. Minimum response time.

One of the most common algorithms in CPU scheduling is the Round Robin algorithm in which, the ready processes waiting in ready queue, are dispatched sequentially and allocate the processor for certain period of time known as time quantum (q) or time slice. If a process is finished during its time quantum, releases the processor, otherwise the processor is pre-empted by the operating system and is allocated to the next ready process waiting in front of the ready queue and the current process will be moved to the end of the this queue. The algorithm continues until all the processes are terminated, or the system is switched off (Silberschatz *et al*, 2006; Willam, 2012).

The most important parameter in this scheduling algorithm (i.e. Round Robin), which highly affects its efficiency, is determining the amount of the time quantum. If a small value is assigned to this parameter, the short processes will pass the system very soon, and the overhead of the system will increase due to high number of context switches. On the other hand, if the amount of the time quantum is larger than the maximum burst time of the ready processes, this policy will downgrade to First Come First Serve scheduling algorithm (Abbas *et al.*, 2011; Soraj and Roy, 2012; Abdulrazaq *et al.*, 2014). The value of this parameter should be taking very important. It is obvious that the amount of this parameter, highly affects the value of waiting time, turnaround time and response time of all the processes, and the number of context switches.

The importance of the problem has already raised the attention of the researchers and study in this field still continues. Some more important works are listed in the following sections.

2.2 Review of Round Robin Scheduling Algorithms

This work has been carried out into two different categories:

1. Scheduling algorithms that statically provide the time quantum in the Round Robin scheduling algorithm and
2. Scheduling algorithms that dynamically determine the time quantum in the Round Robin scheduling algorithm

2.2.1 Static provision of time quantum in Round Robin scheduling algorithm

This section reviews the work done by the various researchers in the field of Round Robin scheduling algorithm by using static (fixed) time quantum.

Ajit *et al* (2010) developed an algorithm and proved the experimental results of its performance over simple Round Robin scheduling algorithm. This algorithm reduces the number of context switching, average waiting time and average turnaround time. The algorithm performs by allocating the CPU to every process in Round Robin fashion with an initial time quantum (say k units). After completing first cycle, it doubles the initial time quantum (2k units); selects the shortest process from the waiting queue and assign the CPU to it and after that, it selects the next shortest process for execution by excluding the already executed process. After completing this cycle, if any process remains in the ready queue after doubling the time quantum, it will half the doubled time quantum (i.e. the initial time quantum) and apply it to the processes in the ready queue. And again, doubles the time quantum if any process remains in the ready queue. This algorithm assumes that all processes arrive at the time in the ready queue.

Ishwari and Deepa (2012) developed, analyzed the operation and performance of the Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems over the simple Round Robin scheduling algorithm and simple priority scheduling algorithm. The algorithm is more efficient because it has less average waiting time, average turnaround time and number of context switches as compared to simple Round Robin, in turn reducing the operating system overhead and dispatch latency. Also, it reduces the problem of starvation as the processes with less remaining CPU burst time are assigned with the higher priorities and are executed first in the second round of algorithm. 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. This algorithm assumes that all processes arrive at the same time in the ready queue.

Manish and AbdulKadir (2012) developed an algorithm that describes an improvement in Round Robin scheduling algorithm. After improvement in Round Robin scheduling algorithm, it was found that the waiting time and turnaround time have been reduced drastically. The Improved Round Robin (IRR) CPU scheduling algorithm works similar to Round Robin (RR) with a small improvement. Improved Round Robin scheduling algorithm picks the first process from the ready queue and allocates 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 is 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 placed at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue. Basically, this thesis will modify this algorithm.

2.2.2 Observations in surveyed literatures

From the literature survey of static provision of time quantum in Round Robin scheduling algorithm the following observations have been drawn:

1. Static method of providing time quantum in Round Robin scheduling algorithm did not have complete knowledge regarding the processes because it is provided by assumptions, between 10-100 milliseconds range.
2. The choice of time quantum if rightly done will produce a very good result in terms of average waiting time, average turnaround time and number of context switch.

2.2.3 Dynamic determination of time quantum in Round Robin scheduling algorithm

This section reviews the work done by various researchers in the field of Round Robin scheduling algorithm by using dynamic time quantum.

Behera *et al* (2010) developed an algorithm and proved the experimental results of its performance over simple Round Robin scheduling algorithm. This algorithm reduces the number of context switching, average waiting time and average turnaround time. 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 followed. 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 rearrange 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. This approach is similar to the approach that will be adopted in this thesis work in the sense that it calculates the time quantum dynamically and applies it to the processes in the ready queue which are arranged in some order.

Samih *et al* (2010) presented an algorithm called the Changeable Time Quantum (CTQ), which determines the time quantum that will be used in Round Robin scheduling algorithm. It does this by using the integer programming (i.e. integer linear programming) method to solve equations that depends on the parameters of the processes (i.e. the number of processes, the burst time of each process, the arrival time of each process, etc) to decide a value (quantum time) that is neither too large nor too small such that every process has got reasonable response time, and the throughput of the system is not decreased due to unnecessary context switches. This method depends on changing time quantum in each round over the cyclic queue. The algorithm was simulated together with the simple Round Robin scheduling algorithm. The burst time and arrival time of the processes were generated using uniform and Poisson distribution respectively. From the simulation, it was observed that the performance of CTQ policy is higher than that of Round Robin in general purpose systems. This approach is similar to the approach that will be adopted in this thesis work in the sense that the burst time and arrival time of the processes will be generated using uniform and Poisson distribution too.

Rakesh *et al* (2011) analyzed the operation and performance of their developed (Fittest Job First Dynamic Round Robin (FJFDRR)) scheduling algorithm over the Priority Based Static Round Robin (PBSRR) scheduling algorithm. Generally, with every process, three factors are associated. These factors are user priority, burst time and arrival time. These factors play an important role in deciding which sequence the processes will be executed. Sorting according to the importance of these factors, user priority comes first, then the burst time and at last the arrival time of the processes. These three factors were mixed up to calculate a new factor i.e. Fit Factor "f" which will decide the order of execution of the processes. But in the algorithm, "f" was calculated for each process. The process having the lowest f value will be scheduled first. The two important criteria that decide the early execution of processes are – higher user priority and shorter burst time. As user priority has higher importance than other factors, so it is given a weight age of 60% and burst time is given 40%, assuming that all the processes have same arrival time i.e. arrival time = 0. Let 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

$$f = UP * UW + SP * BW \quad 2.1$$

Dynamic time quantum is used in order to overcome the limitations of static Round Robin scheduling. To get the optimal time quantum, median of the remaining burst time is taken as the time quantum. This algorithm cannot be used in real time systems because it assumes that all processes arrive at the same time t=0.

Lalit *et al* (2011) developed, analyzed the operation and performance of Optimized Scheduling Algorithm over the simple Round Robin scheduling algorithm. The algorithm arranges the processes in ascending order of the burst time, and then it calculates the time quantum for Round

Robin by taking the average of the burst times. This algorithm assumes that all processes arrive at the time t=0. This approach is similar to the approach that will be adopted in this thesis work in the sense that the time quantum will be calculated in similar manner.

Soraj and Roy (2012) presented a new algorithm called "Adaptive Round Robin Scheduling using Shortest Burst Approach Based on Smart Time Slice", which was discussed in detail, tested and verified. This algorithm gave better results compared to Simple Round Robin Scheduling Algorithm. That is, it reduced the average waiting time, context switches, turnaround time, and throughput of simple Round Robin Scheduling algorithm. The algorithm is a Priority Driven scheduling algorithm based on burst time of processes. First of all, it arranges the processes according to the execution time/burst time in increasing order that is the smaller the burst time, the higher the priority of the running process. The next idea of this approach is to choose the smart time slice (STS), which is mainly dependant on the number of processes. The smart time slice is equal to the burst time of the mid process, when the number of processes is odd. If the number of processes is even, then the smart time slice (STS) will be the average of the CPU burst of all running processes. Based on the experiments and calculations, the algorithm radically solves the fixed time quantum problem which is considered a challenge for Round Robin scheduling algorithm. This algorithm assumes that all processes arrive at the same time in the ready queue.

Saeidi and Hakimeh (2012) presented an algorithm which determined the Optimum Time Quantum Value in Round Robin process scheduling method, the operation and performance of the algorithm were analyzed over some algorithms in the literature of the work and also over the simple Round Robin scheduling algorithm. It does this by using a new non-linear mathematical model which calculates the optimum value of the time quantum in Round Robin process scheduling algorithm, in order to minimize the average waiting time of the processes. This new non-linear mathematical model uses; the total number of ready processes, burst, turnaround, waiting time of the processes etc as its parameters. The model was implemented in Lingo 8.0 software and four selected problems adopted from the literature of the work were implemented and solved. In all the selected examples, the algorithm gives utterly better or equal performance than the previous algorithms in the literature.

Behera *et al* (2012) presented a new algorithm known as A New Proposed Round Robin with Highest Response Ratio Next (RRHRRN) scheduling algorithm for Soft Real Systems, which uses Highest Response Ratio (HRR) criteria for selecting processes from Ready Queue, the operation and performance of the algorithm were analyzed over the algorithm by Behera *et al* (2010) (A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis) in terms of reducing the number of context switches, average waiting time and average turnaround time. It does this by using Highest Response Ratio Next (HRRN) in conjunction with Round Robin (RR). It is similar to Round Robin as processes are added to the tail of the Ready Queue according to arrival time. And after taking the CPU for a time quantum, the process is added again to the tail of the Process Queue, but selection of the processes from the ready queue is based on HRRN criteria i.e. CPU is assigned to the process having highest response ratio by using equation 2.2. The algorithm takes dynamic time quantum into account.

$$Response\ Ratio = \frac{waiting\ time + burst\ time}{burst\ time} \qquad 2.2$$

It finds the dynamic time quantum by taking the mean of burst time of the processes and fills the Ready Queue according to arrival time. The algorithm calculates the Response ratio of each process and assigns the CPU to the process with Highest Response Ratio. After the process is being assigned to the CPU again the Response Ratio is calculated with the updated waiting time of the processes. This loop is continued until all the processes are being executed by the CPU. The dynamic time quantum is computed by taking the mean of the remaining burst time. The processes with shorter burst time and higher waiting time are executed first resulting in better turnaround time and waiting time. This approach is similar to the approach that will be adopted in this thesis in the sense that a process queue will be introduced where newly arriving processes will wait if there are processes in the ready queue waiting for the CPU and processes are added to the tail of the process queue after taking the CPU for a time quantum.

Supriya *et al* (2012) developed an algorithm to determine the optimum time quantum value in Round Robin scheduling, based on vague and non-precise parameters using Linguistic variables and Mamdani Fuzzy logic and called it Fuzzy Interface System for Optimum Time Quantum. It does this by sorting the processes in increasing order of their burst times so that shortest process will be removed earlier from the ready queue to give better turnaround time and waiting time. Whenever a process comes, the required burst time is compared with the available processes in the ready queue and accordingly, the ready queue is updated. The time quantum is taken from the designed Fuzzy Interface System (FIS). This dynamic time quantum is used by all processes present in ready queue and this continues up to complete execution until a new process arrives in the ready queue. In succeeding cycles if new process arrives then time quantum is again calculated taking into consideration of all the parameters. The authors used Matlab software to implement the algorithm and fuzzy interface system. By numerical analysis based on LRRTQ results on the algorithm, it shows that there is an improvement in the performance of the system by reducing unnecessary context switches and also by providing reasonable turnaround time compared to the simple Round Robin scheduling algorithm.

Dibyendu and Iti (2013) developed an algorithm (Dynamic Time Quantum in Shortest Job First Scheduling Algorithm (DTQSJF)) that predicts burst times of processes as in the case of Shortest Job First scheduling and then use the predicted burst times to determine the time quantum to be used in Round Robin scheduling of processes. The algorithm gets the time quantum by taking the average of the predicted burst times and applying the time quantum to the processes in the ready queue in Round Robin fashion. If processes remain in ready queue after completion of their predicted burst time, a new quantum will be generated dynamically depending on another prediction of burst times of remaining processes, and each non completed process will get that amount of time quantum to execute in CPU, if any process does not complete its execution after application of that amount of time (Time Quantum), it will go to tail of the ready queue and wait for its turn. This algorithm was not demonstrated with examples or implemented, but the author suggested that implementing it will decrease the possibilities of starvation.

Surendra (2013) developed, analyzed the operation and performance of "Improving Shortest Remaining Burst Round Robin (ISRBRR) algorithm" over the simple Round Robin scheduling algorithm and Shortest Remaining Burst Round Robin (SRBRR). By experimental analysis it shows that the algorithm performs better than the simple Round Robin and Shortest Remaining Burst Round Robin (SRBRR) algorithm in terms of reducing the number of context switches, average waiting time and average turnaround time. The algorithm performs by assigning the processor to processes with shortest remaining burst in Round Robin manner using the best possible time quantum. Time quantum is computed with the help of median and highest burst time as shown in equation 2.3:

$$tq = \left\lceil \sqrt{median * highest\ burst\ time} \right\rceil \qquad 2.3$$

This algorithm assumes that all processes arrive at the same time in the ready queue.

2.2.4 Observations in surveyed literatures

From the literature survey of dynamic determination of time quantum in Round Robin scheduling algorithm the following observations have been drawn:

1. A dynamic determination of time quantum in Round Robin scheduling algorithm has complete knowledge of the currently active set of processes, but the arrival of new processes may occur in the future, not known to algorithm at the time it is scheduling the current set of processes.
2. A dynamic determination of time quantum in Round Robin scheduling algorithm performs its task (i.e. determine time quantum) during run time and it helps in the CPU utilization.

2.3 Summary of General Observations of Surveyed Literatures in Both Categories

From the literature survey, the following observations have been drawn:

1. Static method of providing time quantum in Round Robin scheduling algorithm did not have complete knowledge regarding the processes because it is provided by assumptions, between 10-100 milliseconds range.
2. The choice of time quantum if rightly done will produce a very good result in terms of average waiting time, average turnaround time and number of context switch.
3. A dynamic determination of time quantum in Round Robin scheduling algorithm has complete knowledge of the currently active set of processes, but the arrival of new processes may occur in the future, not known to algorithm at the time it is scheduling the current set of processes.
4. The dynamic determination of time quantum in Round Robin scheduling algorithm performs its task (i.e. determine time quantum) during run time and it helps in the CPU

utilization by reducing the average waiting time, turnaround time and number context switch.

It has been observed that a lot of works has been done on dynamic determination of time quantum in Round Robin scheduling algorithm, but still there is a scope to design and implement a scheduling algorithm which can increase CPU utilization using dynamic determination of time quantum. Furthermore, the algorithm will also provide efficient results for processes in real time environment.

2.4 The Longest Job First with Combinational Burst Time (LJF+CBT) CPU Scheduling Algorithm

This algorithm by Abdullahi and Junaidu (2013) made an improvement in the Longest Job First (LJF) CPU scheduling algorithm by adding a Combinational Burst Time (CBT) algorithm to it. It assumes that all processes arrived at the same time (i.e. arrival time = 0). It works by sorting the processes in descending order of their burst times. A threshold known as Combined Weighted Average (CWA) given by:

$$\text{CWA} = \frac{\sum_{i=1}^{n} bt[i]}{n}, where\ bt[i] is\ burst\ time\ of\ process\ i \qquad 2.4$$

is defined, which is used to categorized the processes into long and short processes. Long process is a process with burst time greater than CWA; and short process is a process with burst time less or equal to CWA. New burst times are created from this categorization. This is done by merging two consecutive shorter processes in the categorization until no shorter process has one to merge with or no shorter process exist in the categorization. After the merging, new queue is created by sorting the categorized and merged processes in descending order of their burst times. The CPU is then allocated to the processes based on Longest Job First.

2.5 Queuing Theory

A queue is any service or line that receives customers and services them. Queuing theory is the mathematical method of analyzing the congestions and delays of a queue. (Prashant and Gopalkrushna, 2012). Little (1961) describes the relationship between through put rate (i.e. arrival and service rate), cycle time and work in process (i.e. number of customers/jobs in the system). This relationship has been shown to be valid for a wide class of queuing models. The theorem states that the expected number of customers (N) for a system in steady state can be determined using the equation 2.5:

$$L = \lambda T \qquad 2.5$$

where, λ is the average customer arrival rate and T is the average service time for a customer. Consider the example of a restaurant where the customer's arrival rate (λ) doubles but the customers still spend the same amount of time in the restaurant (T). These facts will double the number of customers in the restaurant (L). By the same logic, if the customer arrival rate (λ) remains the same but the customers service time doubles this will also double the total number of customers in the restaurant. This indicates that in order to control the three variables, managerial decisions are only required for any two of the three variables. Three fundamental relationships can be derived from Little's theorem (Prashant and Gopalkrushna, 2012):

1. L increases if λ or T increases
2. λ increases if L increases or T decreases
3. T increases if L increases or λ decreases

2.5.1 Basic Structure of Queuing Models

There are many types of queues; Figure 2.3 depicts an example of a queuing system consisting of a single server model

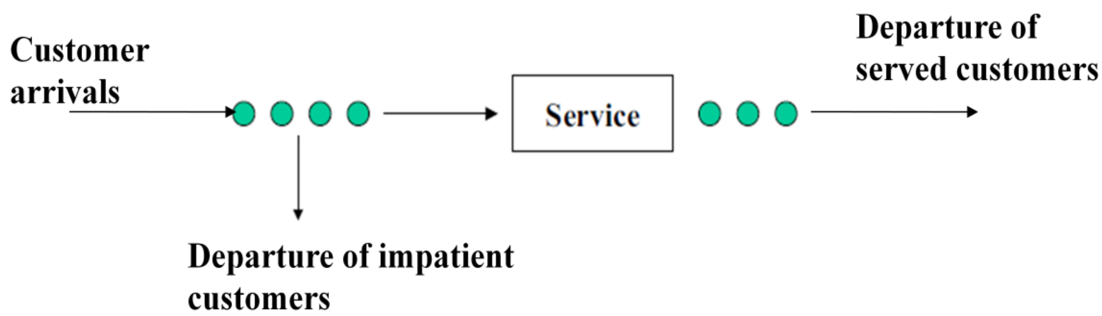


Figure 2.3: The Basic Queuing system (www.emse.fr/~xie/SJTU/Ch6Queue.ppt)

The basic queuing system consists of two major components. Customers arriving at a queuing system wait in queue to get some service. If the system is idle or empty, the arriving customer may be serviced immediately but if the queue is not empty, the customer will have to wait in the queue to get the service. In the course of waiting, if the customer is impatient because of long time taken in the queue, the customer may leave. Once the customer received the service, the customer will leave the system.

2.5.2 Characteristics of queuing systems

Among others, a queuing model is characterized by (Virtamo, 2014; Ivo and Jacques, 2002; www.emse.fr/~xie/SJTU/Ch6Queue.ppt):

1. **The arrival process of customers**

Usually, it is assumed that the inter-arrival times are independent and have a common distribution. In many practical situations customers arrive according to a Poisson stream.

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.

2. The behaviour of customers

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

3. The service times

Usually, it is assume that the service times are independent and identically distributed, and that they are independent of the inter-arrival times. For example, the service times can be deterministic or exponentially distributed. It can also occur that service times are dependent of the queue length. For example, the processing rates of the machines in a production system can be increased once the number of jobs waiting to be processed becomes too large.

4. The service discipline

Customers can be served one by one or in batches. There are many possibilities for the order in which they enter service:

a. first come first served, i.e. in order of arrival;
b. random order;
c. last come first served (e.g. in a computer stack or a shunt buffer in a production line);
d. priorities (e.g. rush orders first, shortest processing time first);
e. processor sharing (in computers that equally divide their processing power over all jobs in the system).

5. **The service capacity**

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

6. **The waiting room**

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

7. **Other factors (in real life).**
    a. screening of the customers
    b. bribing etc
8. **Information available**
    a. upon choice of a queue, does one know the lengths of queues, the service times of individual customers

2.5.3 Kendall Classification of Queuing Systems

Based on the above characteristics of queuing systems in section 2.5.2, there can be different queuing models. These characteristics can be represented with the help of Kendall's notation (www.nptel.ac.in/courses/110106046/Module%209/Lecture%202.pdf).

Kendall (1953), proposed a notation system to represent the six out of the characteristics discussed in section 2.5.2 above. The notation of a queue is written as:

$$A/B/P/Q/R/Z \qquad 2.6$$

where A, B, P, Q, R and Z describe the queuing system properties (Prashant and Gopalkrushna, 2012).

A describes the distribution type of the inter arrival times.

B describes the distribution type of the service times.

P describes the number of servers in the system.

Q (optional) describes the maximum length of the queue.

R (optional) describes the size of the system population.

Z (optional) describes the queuing discipline.

Arrival time distribution (A): Inter-arrival times most commonly fall into one of the following distribution patterns: a Poisson distribution, a Deterministic distribution, or a General distribution. However, inter-arrival times are most often assumed to be independent and memory less, which is the attributes of a Poisson distribution.

Service time distribution (B): The service time distribution can be constant, exponential, hyper exponential, hypo-exponential or general. The service time is independent of the inter-arrival time.

Number of servers (P): The queuing calculations change depends on whether there is a single server or multiple servers for the queue. A single server queue has one server for the queue. This is the situation normally found in a grocery store where there is a line for each cashier. A multiple server queue corresponds to the situation in a bank in which a single line waits for the first of several tellers to become available.

Queue Lengths (optional) (Q): The queue in a system can be modeled as having infinite or finite queue length.

System capacity (optional) (R): The maximum number of customers in a system can be from 1 up to infinity. This includes the customers waiting in the queue.

Queuing discipline (optional) (Z): There are several possibilities in terms of the sequence of customers to be served such as FIFO (First In First Out, i.e. in order of arrival), random order, LIFO (Last In First Out, i.e. the last one to come will be the first to be served), or priorities.

These symbols are used for arrival and service patterns:

M is the Poisson (Markovian) process with exponential distribution of intervals or service duration respectively.

Em is the Erlang distribution of intervals or service duration.

D is the symbol for deterministic (known) arrivals and constant service duration.

G is a general (any) distribution.

GI is a general (any) distribution with independent random values.

Examples (staff.um.edu.mt/jskl1/simweb/intro.htm; Prashant and Gopalkrushna, 2012; www.emse.fr/~xie/SJTU/Ch6Queue.ppt):

D/M/1 = Deterministic (known) input, one exponential server, one unlimited FIFO or unspecified queue, unlimited customer population.

M/G/3/20 = Poisson input, three servers with any distribution, maximum number of customers 20, unlimited customer population.

D/M/1/LIFO/10/50 = Deterministic arrivals, one exponential server, queue is a stack of the maximum size 10, total number of customers 50.

CHAPTER THREE

AN IMPROVED ROUND ROBIN CPU SCHEDULING ALGORITHM

This chapter presents the proposed Improved Round Robin CPU scheduling algorithm, its Pseudo code and flowchart. It also shows how it works with the aid of an illustrative example.

3.1 The Proposed New Improved Round Robin (NIRR) Scheduling Algorithm

The proposed NIRR CPU scheduling algorithm is the modification of the Improved Round Robin (IRR) CPU scheduling algorithm (i.e. the algorithm by Manish and AbdulKadir (2012)). It introduced 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.

The algorithm takes to the REQUEST queue, the first process (i.e.$pr[1]$) that enters the ARRIVE queue, and allocates the CPU to it for the period of its burst time (i.e.$bt[1]$). Processes that arrive while the CPU is executing this process will be added to the ARRIVE queue according to their arrival time. After execution of the first process, all the processes in the ARRIVE queue will be moved to the REQUEST queue and arranged in ascending order of burst times. The algorithm takes the ceiling of the average of burst times of the processes in the REQUEST queue as the time quantum and allocates the CPU to first process in REQUEST queue for the period of the determined time quantum.

$$time\ quantum = \left\lceil \frac{\sum_{i=1}^{n} burst\ time[i]}{n} \right\rceil \qquad 3.1$$

When the time quantum for the process expires, the algorithm checks the remaining CPU burst time of the currently running process. If the remaining CPU burst time is less than or equal to half of the time quantum, the CPU will again be allocated to the currently running process for the remaining CPU burst time. In this case, this process will finish its execution and will be removed from the REQUEST queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than half of the time quantum, the process will be moved to the ARRIVE queue. The CPU scheduler will then proceed to the next process in the REQUEST queue. During the execution of the processes in the REQUEST queue, any process that arrives the system will be placed in the ARRIVE queue. These activities continue until no process is available in the REQUEST queue.

After the execution of the processes in the REQUEST queue; the transferred processes from the REQUEST queue to the ARRIVE queue in the previous execution cycle and the newly arrived processes in the ARRIVE queue will be moved to the REQUEST queue in ascending order of burst times and a new time quantum will be calculated (i.e. the ceiling of the average of burst times of the processes). The CPU will be allocated to the processes in the REQUEST queue as usual using the newly determined time quantum. These activities continue until no process is available in the REQUEST and ARRIVE queues.

3.1.1 The pseudo code of the proposed Round Robin CPU scheduling algorithm

Step 1: Start

Step 2: Make an arrival queue say ARRIVE; where processes will be placed when they arrive the system before they are being moved to the ready queue

Step 3: Make a ready queue say REQUEST

Step 4: WHILE (ARRIVE != NULL)

Step 5: If ($process\_Index == 1$)

$time\_quantum = burst\_time[process\_Index]$

Move the first process ($pr[process\_Index]$) to REQUEST queue

Else

Move all processes in ARRIVE queue to REQUEST queue in ascending order of their burst time

$$time\_quantum = \left\lceil \frac{\sum_{i=1}^{n} burst\ time[i]}{n} \right\rceil \qquad 3.2$$

Step 6: END If

Step 7: WHILE (REQUEST != NULL)

Step 8: Allocate the CPU to the first process in REQUEST queue for a period of 1 time quantum.

Step 9: If the remaining CPU burst time of the currently running process is less than or equal to half time quantum then allocate the CPU again to the currently running process for remaining CPU burst time. After completion of execution, remove the process from the ready queue and go to step 8.

Step 10: If the remaining CPU burst time of the currently running process is longer than half time quantum, remove the process from the REQUEST queue and put it in the ARRIVE queue and go to step 8.

Step 11: If a new process arrives the system, it is placed in the ARRIVE queue.

Step 12: END WHILE.

Step 13: END WHILE.

Step 14: Calculate AWT, ATAT, ART and NCS.

Step 15: END

3.1.2 The flow chart

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

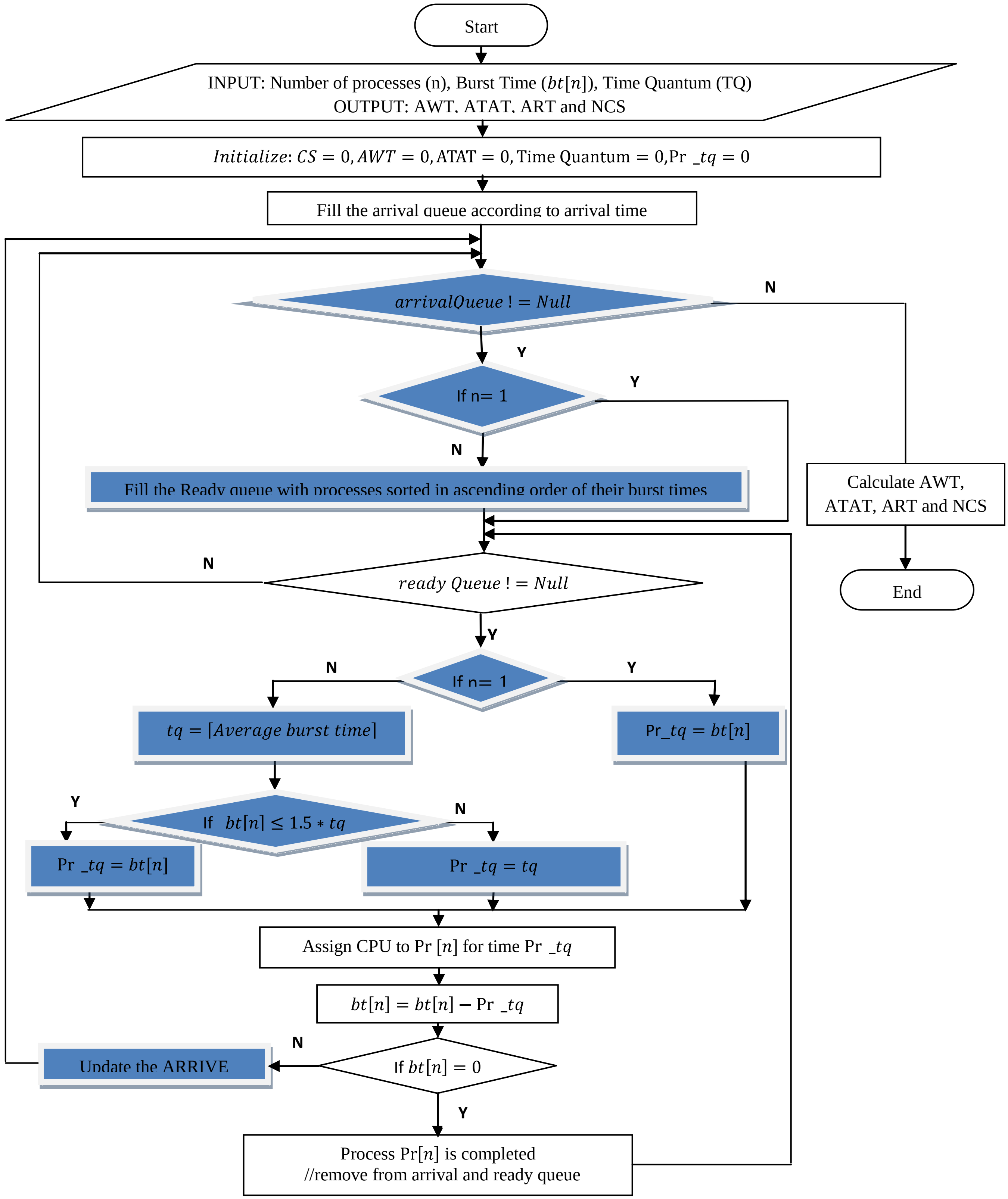


Figure 3.1: The Flow Chart of the Proposed Round Robin CPU Scheduling Algorithm

3.2 How the Proposed Algorithm works

Following all steps of the proposed algorithm stated in the sections 3.1.1, the example that follows will illustrate how the algorithm works. Given 20 processes with their arrival and burst times as shown in Table 3.1, the proposed algorithm works as follows

Table 3.1: Process Table

PR_ID	AT	BT
P1	0	10
P2	2	5
P3	4	3
P4	5	2
P5	6	4
P6	8	6
P7	9	10
P8	10	8
P9	12	12
P10	14	11
P11	16	13
P12	18	4
P13	21	5
P14	25	3
P15	26	1
P16	27	2
P17	28	6
P18	30	4
P19	31	7
P20	33	8

As P1 arrive the system, it will be moved to the REQUEST queue and the CPU will be allocated to it. This is because it is the first process that arrives to the system. So, its burst time $bt[1]$ which equals 10*ms* will be used as the time quantum. After it finishes its execution; P2, P3, P4, P5, P6, P7 and P8 must have arrive the system and will be waiting in the ARRIVE queue. At this point, these 7 processes will be moved to the REQUEST queue in ascending order of their burst times (i.e. P4, P3, P5, P2, P6, P8 and P7), and the average of their burst times will be computed (which is 5.4*ms*) and the ceiling value (i.e. 5*ms*) will be taken as the new time quantum. All the processes will be executed at first round of execution (i.e. assigning the CPU to a process for 1 time quantum and if the remaining burst time of the currently running process after the assignment is less or equal to half of the time quantum, the CPU will be reallocated to the process) except for P8 and P7, because their remaining burst time (3 and 5*ms* respectively) are greater than half of the time quantum (i.e. 2.5*ms*). So, P8 and P7 will be moved to the ARRIVE queue. After the execution of these 7 processes, all the remaining 12 processes must have arrived the system in the ARRIVE queue. The 14 processes (i.e. the 12 newly arrived and the two (P8 and P7) processes which could not finish their execution in the previous execution cycle) will be moved to the REQUEST queue based on the ascending order of their burst times (i.e. P15, P16, P8, P14, P12, P18, P7, P13, P17, P19, P20, P10, P9 and P11). A new time quantum will be computed based on the burst times of these 14 processes in the REQUEST queue. This new time quantum will be 6*ms*. All the processes will be executed at first round of execution except for P10, P9 and P11, because their remaining burst time (5, 6 and 7*ms* respectively) are greater than half of the time quantum (i.e. 3*ms*). So, P10, P9 and P11will be moved to the ARRIVE queue. Since, no process is available in the ARRIVE queue, they will be moved to the REQUEST queue in ascending order of their burst times and a new time quantum will be computed (i.e. 6ms). At this point, all the processes will be executed.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

This chapter presents the design and implementation of the modified Improved Round Robin CPU scheduling algorithm. It starts with the assumptions adopted in the system design; followed by the description of how the various scheduling algorithms under study work with the aid of illustrative example. It also states the system requirement and the system architecture. Finally, the system implementation follows; which is tested using processes that vary between 5 and 3000. The results of this implementation are analyzed.

Assumption

Four performance criteria- average waiting time, average turnaround time, average response time and number of context switches- were studied. First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), Improved Round Robin (IRR), Longest Job First with Combinational Burst Time (LJF+CBT) and the proposed NIRR 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. The system was also assumed to have no context switch cost i.e. the context switching time is equal to zero which means there was no context switch overhead incurred in transferring from one process to another.

4.1 Illustrative Examples

To demonstrate the previous considerations, the following example will be considered, in which each process has its burst and arrival time as shown in Table 4.1. The time quantum to be used by RR and IRR is 10*ms*. All processes are considered to arrive at the same time because one of the algorithms under study (LJF+CBT), considers all processes to arrive at the same time (i.e. t=0).

Table 4.1: Process Table

PR_ID	BT	AT
P1	9	0
P2	27	0
P3	14	0
P4	30	0
P5	22	0
P6	19	0
P7	45	0
P8	39	0
P9	7	0
P10	10	0

For evaluation purpose, the formula of Waiting Time (WT) (i.e. (equation 4.1)) is used in calculating the Average Waiting Time (AWT) (i.e. (equation 4.2)) for each scheduling algorithm.

$$Waiting\ Time = Time\ first\ scheduled - Arrival\ Time \tag{4.1}$$

$$Average\ Waiting\ Time = \frac{Sum\ of\ all\ processes\ Waiting\ Time}{Number\ of\ processes} \tag{4.2}$$

The Turnaround Time (TAT) (i.e. (equation 4.3)) is used in calculating the Average Turnaround Time (ATAT) (i.e. (equation 4.4)) for each scheduling algorithm.

$$Turnaround\ Time = Time\ of\ process\ completion - Arrival\ Time \tag{4.3}$$

$$Average\ Turnaround\ Time = \frac{Sum\ of\ all\ processes\ Turnaround\ Time}{Number\ of\ processes} \tag{4.4}$$

And the Response Time (RT) (i.e. equation 4.5) is used in calculating the Average Response Time (ART) (i.e. equation 4.6) for each scheduling algorithm.

$$ResponseTime = Time\ of\ process's\ first\ response - Arrival\ Time \quad 4.5$$

$$Average\ Response\ Time = \frac{Sum\ of\ all\ processes\ Response\ Time}{Number\ of\ processes} \quad 4.6$$

4.1.1 First Come First Serve (FCFS)

Figure 4.1 shows the Gantt chart of First Come First Serve CPU scheduling algorithm

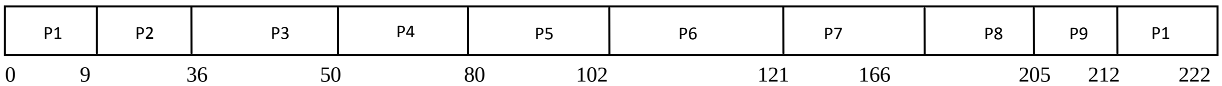


Figure 4.1: Gantt Chart Representation of FCFS

Waiting Time

P1: (0-0) = 0, P2: (9-0) = 9, P3: (36-0) = 36, P4: (50-0) = 50, P5: (80-0) = 80, P6: (102-0) = 102, P7: (121-0) = 121, P8: (166-0) = 166, P9: (205-0) = 205, and P10: (212-0) = 212.

Average Waiting Time

$$AWT = \frac{(0 + 9 + 36 + 50 + 80 + 102 + 121 + 166 + 205 + 212)}{10} = \frac{981}{10} = 98.1$$

Turnaround Time

P1: (9-0) = 9, P2: (36-0) =36, P3: (50-0) = 50, P4: (80-0) =80, P5: (102-0) = 102, P6: (121-0) = 121, P7: (166-0) = 166, P8: (205-0) = 205, P9: (212-0) = 212, P10: (222-0)= 222.

Average Turnaround Time

$$ATAT = \frac{9 + 36 + 50 + 80 + 102 + 121 + 166 + 205 + 212 + 222}{10} = \frac{1203}{10} = 120.3$$

Response Time

P1: (0-0) = 0, P2: (9-0) = 9, P3: (36-0) = 36, P4: (50-0) = 50, P5: (80-0) = 80, P6: (102-0) = 102, P7: (121-0) = 121, P8: (166-0) = 166, P9: (205-0) = 205, and P10: (212-0) = 212.

Average Response Time

$$ART = \frac{(0 + 9 + 36 + 50 + 80 + 102 + 121 + 166 + 205 + 212)}{10} = \frac{981}{10} = 98.1$$

Number of Context Switches = 9

4.1.2 Shortest Job First (SJF)

Figure 4.2 shows the Gantt chart of Shortest Job First CPU scheduling algorithm

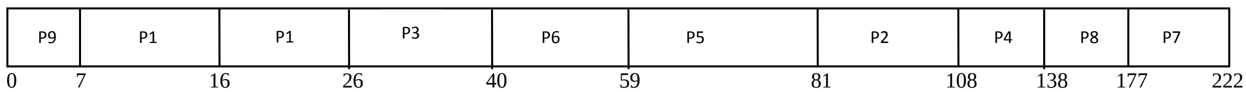


Figure 4.2: Gantt Chart Representation of SJF

Waiting Time

P1: (7-0) = 7, P2: (81-0) = 81, P3: (26-0) = 26, P4: (108-0) = 108, P5: (59-0) = 59, P6: (40-0) = 40, P7: (177-0) = 177, P8: (138-0) = 138, P9: (0-0) = 0, and P10: (16-0) = 16.

Average Waiting Time

$$AWT = \frac{(7 + 81 + 26 + 108 + 59 + 40 + 177 + 138 + 0 + 16)}{10} = \frac{652}{10} = 65.2$$

Turnaround Time

P1: (16-0) = 16, P2: (108-0) =108, P3: (40-0) = 40, P4: (138-0) =138, P5: (81-0) = 81, P6: (59-0) = 59, P7: (222-0) = 222, P8: (177-0) = 177, P9: (7-0) = 7, P10: (26-0) = 26.

Average Turnaround Time

$$ATAT = \frac{16 + 108 + 40 + 138 + 81 + 59 + 222 + 177 + 7 + 26}{10} = \frac{874}{10} = 87.4$$

Response Time

P1: (7-0) = 7, P2: (81-0) = 81, P3: (26-0) = 26, P4: (108-0) = 108, P5: (59-0) = 59, P6: (40-0) = 40, P7: (177-0) = 177, P8: (138-0) = 138, P9: (0-0) = 0, and P10: (16-0) = 16.

Average Response Time

$$ART = \frac{(7 + 81 + 26 + 108 + 59 + 40 + 177 + 138 + 0 + 16)}{10} = \frac{652}{10} = 65.2$$

Number of Context Switches = 9

4.1.3 Round Robin (RR)

Figure 4.3 shows the Gantt chart of Round Robin CPU scheduling algorithm

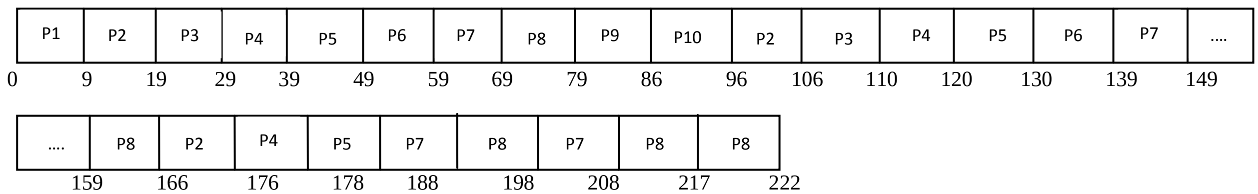


Figure 4.3: Gantt Chart Representation of RR

Waiting Time

P1: (0-0) = 0, P2: ((9+77+53)-0) =139, P3: ((19+77)-0) = 96, P4: ((29+71+46)-0) = 146, P5: ((39+71+46)-0) = 156, P6: ((49+71)-0) = 120, P7: ((59+70+29+10+9)-0) = 177, P8: ((69+70+29+10)-0) = 178, P9: ((79)-0) = 79, P10: (86-0) = 86.

Average Waiting Time

$$AWT = \frac{(0 + 139 + 96 + 146 + 156 + 120 + 177 + 178 + 79 + 86)}{10} = \frac{1177}{10} = 117.7$$

Turnaround Time

P1: (9-0) = 9, P2: (166-0) =166, P3: (110-0) = 110, P4: (176-0) =176, P5: (178-0) = 178, P6: (139-0) = 139, P7: (222-0) = 222, P8: (217-0) = 217, P9: (86-0) = 86, P10: (96-0) = 96.

Average Turnaround Time

$$ATAT = \frac{9 + 166 + 110 + 176 + 178 + 139 + 222 + 217 + 86 + 96}{10} = \frac{1399}{10} = 139.9$$

Response Time

P1: (0-0) = 0, P2: (9-0) = 9, P3: (19-0) = 19, P4: (29-0) = 29, P5: (39-0) = 39, P6: (49-0) = 49, P7: (59-0) = 59, P8: (69-0) = 69, P9: (79-0) = 79, and P10: (86-0) = 86.

Average Response Time

$$ART = \frac{(0 + 9 + 19 + 29 + 39 + 49 + 59 + 69 + 79 + 86)}{10} = \frac{438}{10} = 43.8$$

Number of Context Switches = 24

4.1.4 Improved Round Robin (IRR)

Figure 4.4 shows the Gantt chart of Improved Round Robin CPU scheduling algorithm

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P2	P4	P5	P7	P8	P4	....
0	9	19	33	43	53	72	82	92	99	109	126	136	148	158	168	178

....	P7	P8	P7
	188	207	222

Figure 4.4: Gantt Chart Representation of IRR

Waiting Time

P1: (0-0) = 0, P2: ((9+90)-0) =99, P3: (19-0) = 19, P4: ((33+83+32)-0) = 148, P5: ((43+83)-0) = 126, P6: (53)-0) = 53, P7: ((72+66+20+19)-0) = 177, P8: ((82+66+20)-0) = 168, P9: ((92)-0) = 92, P10: (99-0) = 99.

Average Waiting Time

$$AWT = \frac{(0 + 99 + 19 + 148 + 126 + 53 + 177 + 168 + 92 + 99)}{10} = \frac{981}{10} = 98.1$$

Turnaround Time

P1: (9-0) = 9, P2: (126-0) =126, P3: (33-0) = 33, P4: (178-0) =178, P5: (148-0) = 148, P6: (72-0) = 72, P7: (222-0) = 222, P8: (207-0) = 207, P9: (99-0) = 99, P10: (109-0) = 109.

Average Turnaround Time

$$ATAT = \frac{9 + 126 + 33 + 178 + 148 + 72 + 222 + 207 + 99 + 109}{10} = \frac{1203}{10} = 120.3$$

Response Time

P1: (0-0) = 0, P2: (9-0) = 9, P3: (19-0) = 19, P4: (33-0) = 33, P5: (43-0) = 43, P6: (53-0) = 53, P7: (72-0) = 72, P8: (82-0) = 82, P9: (92-0) = 92, and P10: (99-0) = 99.

Average Response Time

$$ART = \frac{(0 + 9 + 19 + 33 + 43 + 53 + 72 + 82 + 92 + 99)}{10} = \frac{502}{10} = 50.2$$

Number of Context Switches = 18

4.1.5 Longest Job First with Combinational Burst Time (LJF+CBT)

According to the algorithm, it sorts, merges and resorts the burst times of the processes as shown in Tables 4.2 and 4.3.

Table 4.2: Processes Sorted in Decreasing Order of Burst Time

PR_ID	BT
P7	45
P8	39
P4	30
P2	27
P5	22
P6	19
P3	14
P10	10
P1	9
P9	7

Table 4.3: Longer Jobs + CBT

PR_ID	BT
P7	45
*P5,6	41
P8	39
P4	30
P2	27
*P3,10	24
*P9,1	16

Figure 4.5 shows the Gantt chart of LJF+CBT CPU scheduling algorithm

P7	*P5, 6	P8	P4	P2	*P6, 10	*P1, 9

0 45 86 125 155 182 206 222

Figure 4.5: Gantt Chart Representation of LJF+CBT

Waiting Time

*P1, 9: (206-0) = 206, where *P1, 9 is the result of merging P1 and P9, at Time = 0 and thus had to wait for the longer processes to proceed. P2: (155-0) = 155, where P2 arrived at Time = 0 and scheduled at 155. *P3, 10: (182-0) = 182, P4: (125-0) = 125, *P5, 6: (45-0) = 45, P7: (0-0) = 0, P8: (86-0) = 86

Average Waiting Time

$$AWT = \frac{(206 + 155 + 182 + 125 + 45 + 0 + 86)}{10} = \frac{799}{10} = 79.9$$

Turnaround Time

*P1, 9: (222-0) = 222, P2: (182-0) =182, *P3, 10: (206-0) =206, P4: (155-0) =155, *P5, 6: (86-0) = 86, P7: (45-0) = 45, P8: (125-0) = 125.

Average Turnaround Time

$$ATAT = \frac{(222 + 182 + 206 + 155 + 86 + 45 + 125)}{10} = \frac{1021}{10} = 102.1$$

Response Time

*P1, 9: (206-0) = 206, P2: (155-0) = 155, *P3, 10: (182-0) = 182, P4: (125-0) = 125, *P5, 6: (45-0) = 45, P7: (0-0) = 0, P8: (86-0) = 86

Average Response Time

$$ART = \frac{(206 + 155 + 182 + 125 + 45 + 0 + 86)}{10} = \frac{799}{10} = 79.9$$

Number of Context Switches = 6

4.1.6 The Proposed Round Robin CPU scheduling algorithm

Figure 4.6 shows the Gantt chart of the Proposed CPU scheduling algorithm

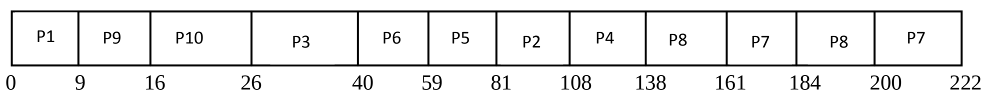


Figure 4.6: Gantt Chart Representation of the Proposed Algorithm

Waiting Time

P1: (0-0) = 0, P2: (81-0) = 81, P3: (26-0) = 26, P4: (108-0) = 108, P5: (59-0) = 59, P6: (40-0) = 40, P7: ((161+16)-0) = 177, P8: ((138+23)-0) = 161, P9: (9-0) = 9, and P10: (16-0) = 16.

Average Waiting Time

$$AWT = \frac{(0 + 81 + 26 + 108 + 59 + 40 + 177 + 161 + 9 + 16)}{10} = \frac{677}{10} = 67.7$$

Turnaround Time

P1: (16-0) = 9, P2: (108-0) =108, P3: (40-0) = 40, P4: (138-0) =138, P5: (81-0) = 81, P6: (59-0) = 59, P7: (222-0) = 222, P8: (177-0) = 177, P9: (7-0) = 7, P10: (26-0) = 26.

Average Turnaround Time

$$ATAT = \frac{9 + 108 + 40 + 138 + 81 + 59 + 222 + 200 + 16 + 26}{10} = \frac{899}{10} = 89.9$$

Response Time

P1: (0-0) = 0, P2: (81-0) = 81, P3: (26-0) = 26, P4: (108-0) = 108, P5: (59-0) = 59, P6: (40-0) = 40, P7: (161-0) = 161, P8: (138-0) = 138, P9: (9-0) =9, and P10: (16-0) = 16.

Average Response Time

$$ART = \frac{(0 + 81 + 26 + 108 + 59 + 40 + 161 + 138 + 9 + 16)}{10} = \frac{638}{10} = 63.8$$

Number of Context Switches = 11

Table 4.4: Comparative Table

Algorithms	AWT	ATAT	ART	NCS
FCFS	98.1	120.3	98.1	9
SJF	65.2	87.4	65.2	9
RR	117.7	139.9	43.8	24
IRR	98.1	120.3	50.2	18
LJF+CBT	79.9	102.1	79.9	6
NIRR	67.7	89.9	63.8	11

Table 4.4 shows the comparative results of the algorithms under study. SJF has the minimal AWT and ATAT while LJF+CBT and RR have the minimal NCS and ART respectively. In the RR category, the proposed NIRR CPU scheduling algorithm has the minimal AWT, ATAT and NCS.

4.2 System Requirements

4.2.1 Experimental setup

Hardware

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

b. 1.5GB of RAM and

c. 75GB of hard disk

Software

a. Window XP operating system

b. NetBeans IDE 6.7.1 version and JDK1.7

A process generator routine was built to generate the process sets. Each process in the process set is a tuple: <(process_*id*, *arrival_time*, *CPU_time*)>. The process arrival was modeled as a Poisson random process. Hence, the inter-arrival times are Poisson distributed. A process arrival generator was developed to take care of the random arrival of different processes to the system. The generator produces the inter-arrival times utilizing some specific mean (arrival intensity) of the distribution function.

Let *F*(x) be a Poisson process with a mean α>0. So, using the Inverse Transformation method:

$$F(x) = \frac{e^{-\alpha}\alpha^{x}}{x!} \qquad 4.7$$

One important properties of the Poisson distribution is that the mean and variance are both equal to α, that is,

$$E(X) = \alpha = V(X) \qquad 4.8$$

Burst time (i.e. the *CPU_time*) was generated using normal, uniform or exponential distribution. A process burst time generator was developed to take care of the random burst time of different processes in the system.

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

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \qquad 4.9$$

Uniform distribution: Consider a random variable X that is uniformly distributed on the interval [a, b]. So, using the Inverse Transformation method:

The probability density function (pdf) is given by:

$$f(x) = \frac{1}{b-a} \tag{4.10}$$

Where $a \le x \le b$

And the cumulative density function (cdf) is given by:

$$F(x) = \frac{x-a}{b-a} = R \tag{4.11}$$

Or

$$x = a + (b-a)R \tag{4.12}$$

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

Its mean and variance is given by:

$$E(X) = \frac{a+b}{2} and\, V(X) = \frac{(b-a)^2}{12} \tag{4.13}$$

Exponential distribution: Consider a random variable X that is exponentially distributed with a parameter rate of exponential distribution λ. The probability density function (pdf) is given by:

$$f(x) = \lambda e^{-\lambda x} \tag{4.14}$$

It has mean and variance is given by:

$$E(X) = \frac{1}{\lambda} and\ V(X) = \frac{1}{\lambda^2} \qquad 4.15$$

4.3 System Architecture

The logical architecture of the system is depicted in Figure 4.7. The system takes in as input the number of processes to be considered, the range (lower and upper bound) of the burst times, the statistical distribution used in generating burst times and the time quantum used by RR and IRR CPU scheduling algorithms. The number of processes specified is generated using the specified statistical distribution (i.e. Normal, Uniform or Exponential) for generating the burst and Poisson distribution for generating arrival times. The generated processes are stored in the process log and are arranged in either arrive or ready queue according to their arrival time 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 criteria for all the algorithms can be drawn.

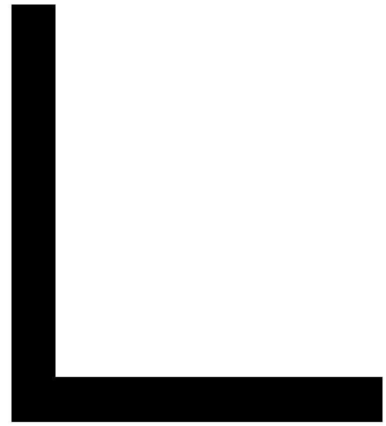


Figure 4.7: The System Architecture

4.4 Discussions

Figure 4.8 shows the system interface through which the user can effectively communicate with the system. It also shows some results of computing 15 processes with burst times that are uniformly distributed ranging between 1 and 200*ms*, and the time quantum of 50*ms* used by RR and IRR.

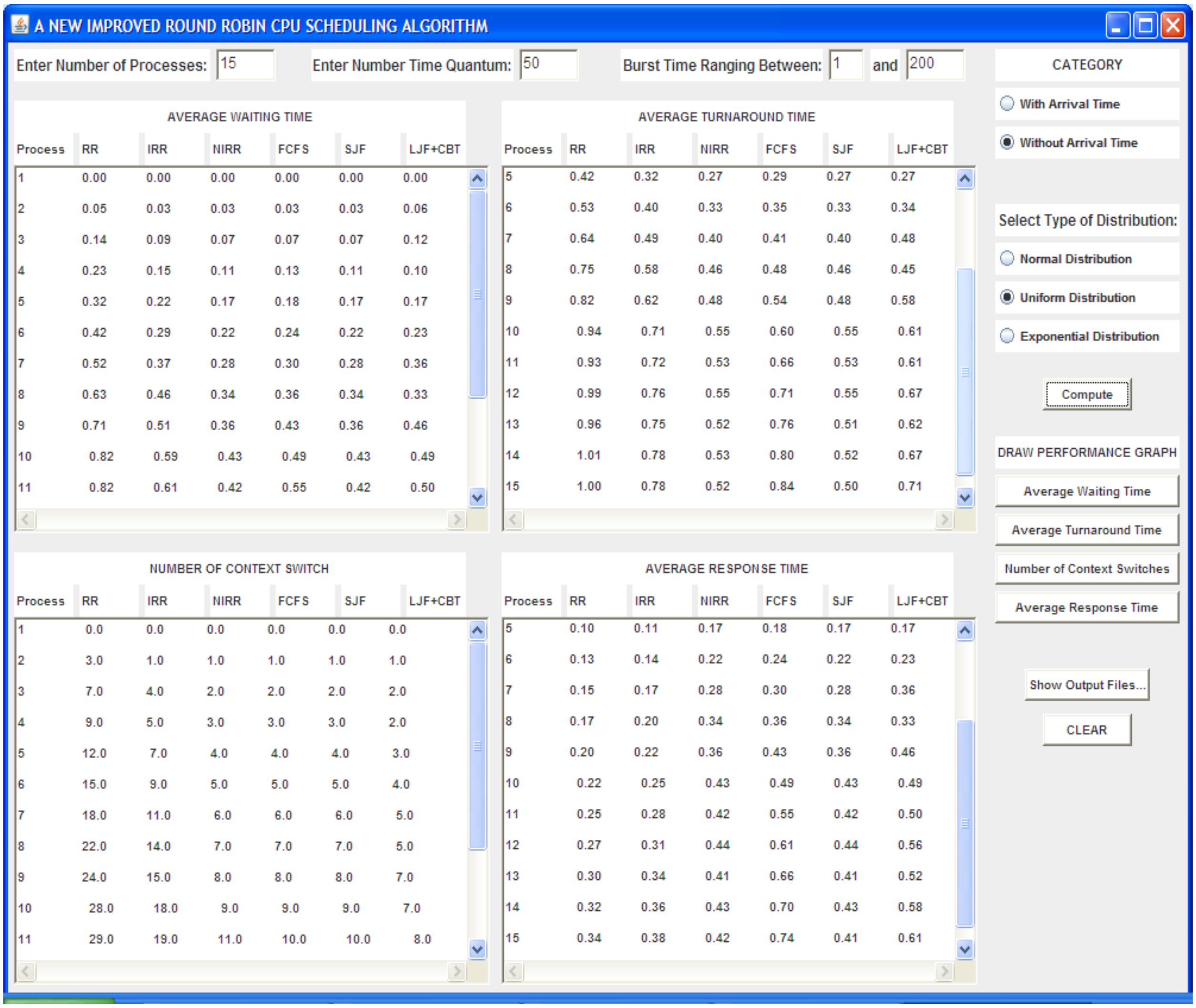
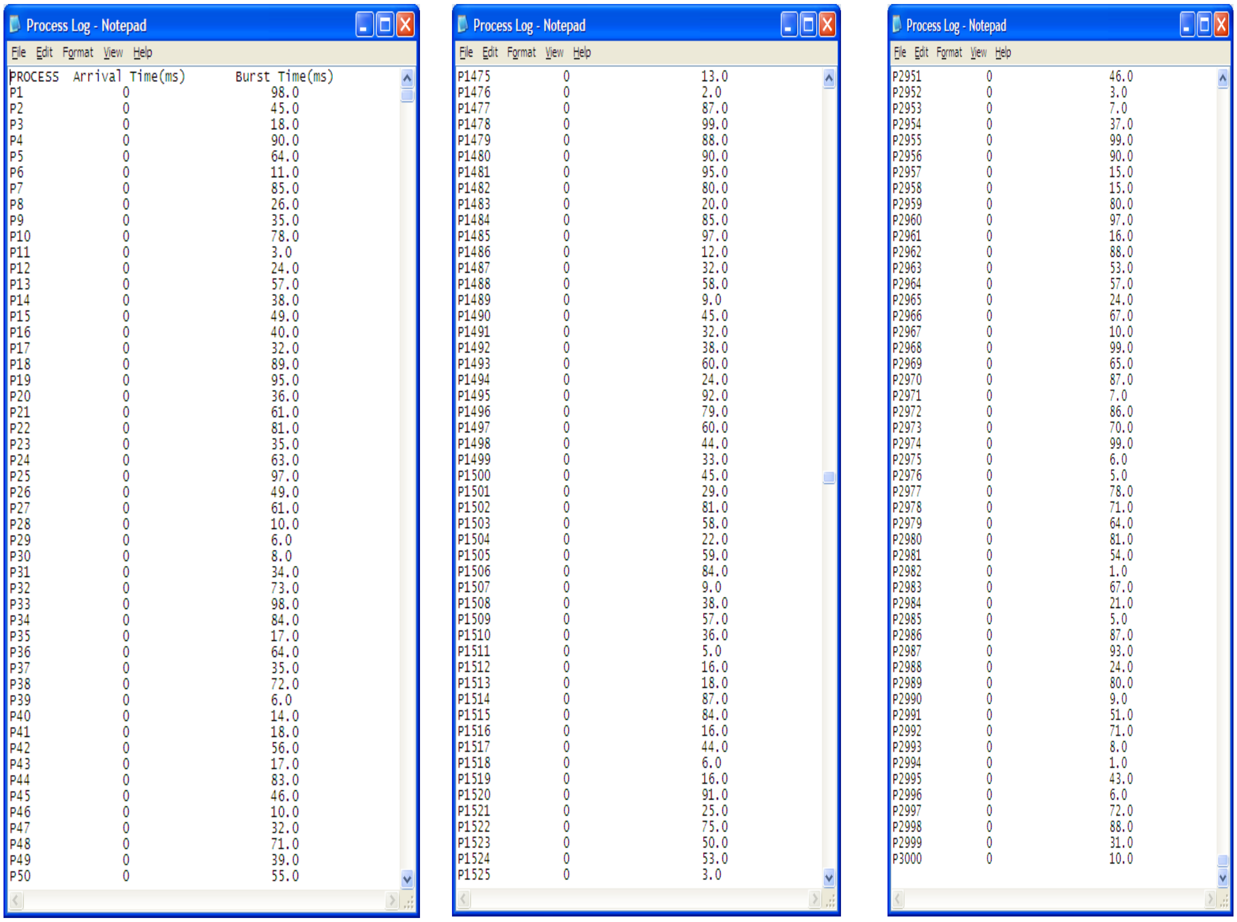


Figure 4.8: System Interface with Some Process Execution in Progress

Figure 4.9 shows a snap shot of some of the 3000 processes (i.e. the ID, Burst time (*ms*) and Arrival time (*ms*)) used in computation.



PROCESS	Arrival Time(ms)	Burst Time(ms)
P1	0	98.0
P2	0	45.0
P3	0	18.0
P4	0	90.0
P5	0	64.0
P6	0	11.0
P7	0	85.0
P8	0	26.0
P9	0	35.0
P10	0	78.0
P11	0	3.0
P12	0	24.0
P13	0	57.0
P14	0	38.0
P15	0	49.0
P16	0	40.0
P17	0	32.0
P18	0	89.0
P19	0	95.0
P20	0	36.0
P21	0	61.0
P22	0	81.0
P23	0	35.0
P24	0	63.0
P25	0	97.0
P26	0	49.0
P27	0	61.0
P28	0	10.0
P29	0	6.0
P30	0	8.0
P31	0	34.0
P32	0	73.0
P33	0	98.0
P34	0	84.0
P35	0	17.0
P36	0	64.0
P37	0	35.0
P38	0	72.0
P39	0	6.0
P40	0	14.0
P41	0	18.0
P42	0	56.0
P43	0	17.0
P44	0	83.0
P45	0	46.0
P46	0	10.0
P47	0	32.0
P48	0	71.0
P49	0	39.0
P50	0	55.0

P1475	0	13.0
P1476	0	2.0
P1477	0	87.0
P1478	0	99.0
P1479	0	88.0
P1480	0	90.0
P1481	0	95.0
P1482	0	80.0
P1483	0	20.0
P1484	0	85.0
P1485	0	97.0
P1486	0	12.0
P1487	0	32.0
P1488	0	58.0
P1489	0	9.0
P1490	0	45.0
P1491	0	32.0
P1492	0	38.0
P1493	0	60.0
P1494	0	24.0
P1495	0	92.0
P1496	0	79.0
P1497	0	60.0
P1498	0	44.0
P1499	0	33.0
P1500	0	45.0
P1501	0	29.0
P1502	0	81.0
P1503	0	58.0
P1504	0	22.0
P1505	0	59.0
P1506	0	84.0
P1507	0	9.0
P1508	0	38.0
P1509	0	57.0
P1510	0	36.0
P1511	0	5.0
P1512	0	16.0
P1513	0	18.0
P1514	0	87.0
P1515	0	84.0
P1516	0	16.0
P1517	0	44.0
P1518	0	6.0
P1519	0	16.0
P1520	0	91.0
P1521	0	25.0
P1522	0	75.0
P1523	0	50.0
P1524	0	53.0
P1525	0	3.0

P2951	0	46.0
P2952	0	3.0
P2953	0	7.0
P2954	0	37.0
P2955	0	99.0
P2956	0	90.0
P2957	0	15.0
P2958	0	15.0
P2959	0	80.0
P2960	0	97.0
P2961	0	16.0
P2962	0	88.0
P2963	0	53.0
P2964	0	57.0
P2965	0	24.0
P2966	0	67.0
P2967	0	10.0
P2968	0	99.0
P2969	0	65.0
P2970	0	87.0
P2971	0	7.0
P2972	0	86.0
P2973	0	70.0
P2974	0	99.0
P2975	0	6.0
P2976	0	5.0
P2977	0	78.0
P2978	0	71.0
P2979	0	64.0
P2980	0	81.0
P2981	0	54.0
P2982	0	1.0
P2983	0	67.0
P2984	0	21.0
P2985	0	5.0
P2986	0	87.0
P2987	0	93.0
P2988	0	24.0
P2989	0	80.0
P2990	0	9.0
P2991	0	51.0
P2992	0	71.0
P2993	0	8.0
P2994	0	1.0
P2995	0	43.0
P2996	0	6.0
P2997	0	72.0
P2998	0	88.0
P2999	0	31.0
P3000	0	10.0

Figure 4.9: Process Identity Snap Shot

4.4.1 Results of normal distribution

The following shows the relationships between the CPU scheduling algorithms under study using 3000 processes generated by the normal distribution with a mean of 50.5*ms*, standard deviation of 28.6*ms* and time quantum of 25*ms* used by RR and IRR.

Average Waiting Time - Notepad

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.13	0.11	0.07	0.11	0.07	0.11
10	0.34	0.26	0.17	0.25	0.17	0.23
15	0.52	0.4	0.27	0.37	0.27	0.31
20	0.79	0.62	0.43	0.52	0.42	0.47
25	1.01	0.78	0.55	0.68	0.54	0.57
30	1.06	0.83	0.56	0.83	0.54	0.67
35	1.27	0.98	0.67	0.95	0.65	0.74
40	1.42	1.09	0.75	1.08	0.72	0.88
45	1.54	1.19	0.81	1.2	0.78	0.94
50	1.69	1.31	0.89	1.32	0.86	1.0
55	1.86	1.44	0.98	1.43	0.95	1.14
60	2.08	1.6	1.1	1.55	1.07	1.23
65	2.32	1.78	1.24	1.68	1.21	1.4
70	2.55	1.94	1.36	1.81	1.33	1.54
75	2.69	2.05	1.44	1.94	1.4	1.6
80	2.81	2.15	1.48	2.06	1.44	1.69
85	3.02	2.32	1.61	2.19	1.56	1.79
90	3.15	2.43	1.68	2.31	1.63	1.86
95	3.32	2.56	1.77	2.44	1.71	1.98
100	3.39	2.64	1.8	2.56	1.73	2.15
110	3.79	2.94	2.02	2.81	1.94	2.35
120	4.12	3.2	2.2	3.06	2.11	2.55
130	4.5	3.48	2.39	3.31	2.3	2.75
140	4.9	3.78	2.63	3.57	2.52	3.02
150	5.24	4.05	2.8	3.83	2.69	3.21
160	5.63	4.36	3.01	4.09	2.89	3.49
170	5.98	4.64	3.18	4.35	3.07	3.75
180	6.28	4.87	3.36	4.61	3.21	3.96
190	6.65	5.16	3.55	4.87	3.4	4.23
200	6.99	5.42	3.73	5.13	3.58	4.44
220	7.76	6.02	4.14	5.65	3.99	4.82
240	8.22	6.42	4.36	6.16	4.18	5.24
260	8.98	7.02	4.78	6.67	4.58	5.72
280	9.56	7.49	5.1	7.17	4.86	6.17
300	10.11	7.92	5.37	7.67	5.12	6.48

Average Waiting Time - Notepad

300	10.11	7.92	5.37	7.67	5.12	6.48
320	10.79	8.44	5.73	8.16	5.48	6.88
340	11.49	9.01	6.11	8.65	5.86	7.36
360	12.19	9.54	6.46	9.14	6.23	7.75
380	13.01	10.15	6.91	9.64	6.67	8.24
400	13.79	10.73	7.31	10.14	7.09	8.74
425	14.67	11.43	7.78	10.78	7.55	9.28
450	15.59	12.15	8.28	11.42	8.04	9.79
475	16.54	12.87	8.78	12.06	8.52	10.35
500	17.34	13.48	9.23	12.7	8.92	10.88
550	19.01	14.77	10.1	13.97	9.76	11.89
600	20.71	16.1	11.02	15.24	10.64	13.01
650	22.39	17.41	11.92	16.5	11.51	13.99
700	24.36	18.94	13.0	17.78	12.59	15.16
750	26.3	20.43	14.07	19.08	13.63	16.4
800	28.22	21.9	15.11	20.39	14.64	17.61
850	30.1	23.33	16.04	21.71	15.6	18.8
900	31.83	24.62	16.97	23.02	16.44	19.67
950	33.32	25.82	17.68	24.31	17.16	20.52
1000	35.04	27.12	18.61	25.59	18.04	21.6
1100	38.64	29.86	21.37	28.13	20.72	23.66
1200	42.6	32.85	24.54	30.63	23.81	25.16
1300	45.66	35.28	27.67	33.18	26.84	27.18
1400	49.1	37.92	30.65	35.76	29.74	28.87
1500	52.31	40.4	33.37	38.31	32.45	30.84
1600	55.92	43.08	36.09	40.84	35.09	32.63
1700	59.68	45.94	38.84	43.35	37.71	34.14
1800	63.12	48.59	41.7	45.85	40.49	35.93
1900	66.6	51.26	44.54	48.36	43.25	37.67
2000	69.99	53.92	47.33	50.92	45.95	39.56
2200	76.67	59.13	52.78	55.98	51.27	43.67
2400	83.35	64.35	58.22	61.01	56.57	47.14
2600	90.55	69.78	63.58	66.05	61.72	50.78
2800	97.71	75.3	69.05	71.17	66.99	54.1
3000	105.16	80.99	74.36	76.3	72.07	57.92

Figure 4.10: Results of Average Waiting Times obtained using Normal Distribution

Average Turnaround Time - Notepad

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.19	0.17	0.13	0.17	0.13	0.17
10	0.39	0.31	0.22	0.3	0.22	0.28
15	0.57	0.45	0.33	0.42	0.33	0.36
20	0.85	0.68	0.49	0.58	0.49	0.53
25	1.07	0.84	0.61	0.74	0.6	0.63
30	1.12	0.88	0.61	0.88	0.59	0.72
35	1.33	1.03	0.72	1.01	0.71	0.79
40	1.47	1.14	0.8	1.13	0.77	0.93
45	1.59	1.25	0.86	1.25	0.83	0.99
50	1.74	1.36	0.94	1.37	0.91	1.05
55	1.91	1.49	1.03	1.48	1.0	1.19
60	2.13	1.65	1.15	1.6	1.12	1.28
65	2.37	1.83	1.29	1.73	1.26	1.46
70	2.6	2.0	1.41	1.86	1.38	1.59
75	2.75	2.1	1.49	1.99	1.45	1.65
80	2.86	2.2	1.54	2.11	1.49	1.74
85	3.07	2.37	1.66	2.24	1.61	1.84
90	3.2	2.48	1.73	2.36	1.68	1.91
95	3.37	2.61	1.82	2.49	1.76	2.04
100	3.44	2.69	1.85	2.61	1.78	2.2
110	3.84	2.99	2.07	2.86	1.99	2.4
120	4.17	3.25	2.25	3.11	2.16	2.6
130	4.55	3.53	2.44	3.36	2.35	2.8
140	4.95	3.83	2.68	3.62	2.57	3.07
150	5.29	4.1	2.85	3.88	2.74	3.26
160	5.69	4.41	3.06	4.14	2.95	3.54
170	6.04	4.69	3.23	4.4	3.12	3.8
180	6.33	4.92	3.41	4.66	3.26	4.01
190	6.7	5.22	3.6	4.92	3.46	4.28
200	7.04	5.48	3.78	5.18	3.63	4.49
220	7.81	6.07	4.2	5.7	4.04	4.88
240	8.27	6.47	4.41	6.21	4.23	5.29
260	9.03	7.07	4.83	6.72	4.63	5.77
280	9.61	7.54	5.15	7.22	4.91	6.22
300	10.16	7.97	5.42	7.72	5.17	6.53

Average Turnaround Time - Notepad

300	10.16	7.97	5.42	7.72	5.17	6.53
320	10.84	8.49	5.78	8.21	5.53	6.93
340	11.54	9.06	6.16	8.7	5.91	7.41
360	12.24	9.59	6.51	9.19	6.28	7.8
380	13.06	10.2	6.96	9.69	6.72	8.29
400	13.84	10.79	7.36	10.19	7.14	8.79
425	14.72	11.48	7.83	10.83	7.6	9.33
450	15.64	12.2	8.33	11.47	8.09	9.84
475	16.6	12.92	8.84	12.11	8.57	10.4
500	17.39	13.54	9.28	12.75	8.97	10.93
550	19.06	14.82	10.15	14.02	9.81	11.94
600	20.76	16.16	11.07	15.29	10.69	13.06
650	22.44	17.46	11.97	16.55	11.56	14.04
700	24.41	18.99	13.05	17.83	12.64	15.21
750	26.36	20.48	14.12	19.13	13.68	16.45
800	28.28	21.96	15.17	20.44	14.69	17.66
850	30.15	23.38	16.09	21.76	15.66	18.85
900	31.88	24.68	17.02	23.07	16.49	19.72
950	33.37	25.87	17.74	24.36	17.21	20.57
1000	35.09	27.17	18.66	25.64	18.09	21.65
1100	38.69	29.91	21.42	28.18	20.78	23.71
1200	42.66	32.9	24.59	30.68	23.86	25.21
1300	45.71	35.33	27.73	33.23	26.89	27.24
1400	49.15	37.97	30.7	35.81	29.79	28.92
1500	52.36	40.45	33.42	38.36	32.5	30.89
1600	55.97	43.13	36.14	40.9	35.14	32.68
1700	59.73	45.99	38.89	43.4	37.76	34.19
1800	63.17	48.64	41.75	45.9	40.54	35.98
1900	66.65	51.31	44.6	48.41	43.3	37.72
2000	70.04	53.97	47.39	50.97	46.01	39.61
2200	76.72	59.18	52.83	56.03	51.32	43.72
2400	83.4	64.4	58.28	61.06	56.62	47.19
2600	90.6	69.83	63.63	66.1	61.78	50.83
2800	97.76	75.35	69.1	71.22	67.04	54.15
3000	105.21	81.04	74.42	76.35	72.13	57.97

Figure 4.11: Results of Average Turnaround Times obtained using Normal Distribution

Figures 4.10 and 4.11 respectively present the results of Average Waiting Time and Average Turnaround Time obtained for the different times of runs and different processes generated using normal distribution.

Figure 4.12 shows the overall graphical result of the Average Waiting Time for all cases of values taken. It was observed from the graph that the SJF has the best performance when the number of processes is less than 1500 and the proposed algorithm (NIRR) came second, followed by LJF+CBT, FCFS, IRR and RR respectively. But, when the number of processes exceeds 1500, LJF+CBT tends to produce the best performance, SJF was second, then the proposed algorithm (NIRR) came third, followed by FCFS, IRR and RR respectively.

Figure 4.13 shows the overall graphical result of the Average Turnaround Time for all cases of values taken. It was observed from the graph that the SJF has the best performance when the number of processes is less than 1500 and the proposed algorithm (NIRR) came second, followed by LJF+CBT, FCFS, IRR and RR respectively.. But, when the number of processes exceeds 1500, LJF+CBT tends to produce the best performance, SJF was second, then the proposed algorithm (NIRR) came third, followed by FCFS, IRR and RR respectively.

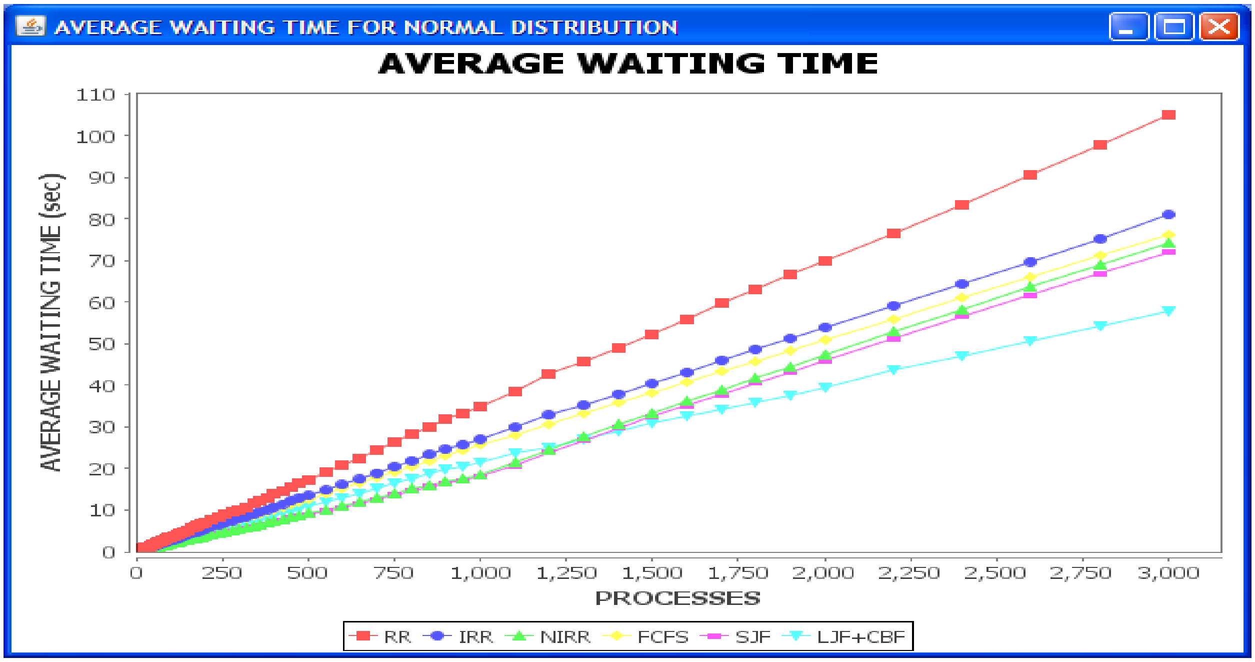


Figure 4.12: Graph of Average Waiting Time using Normal Distribution

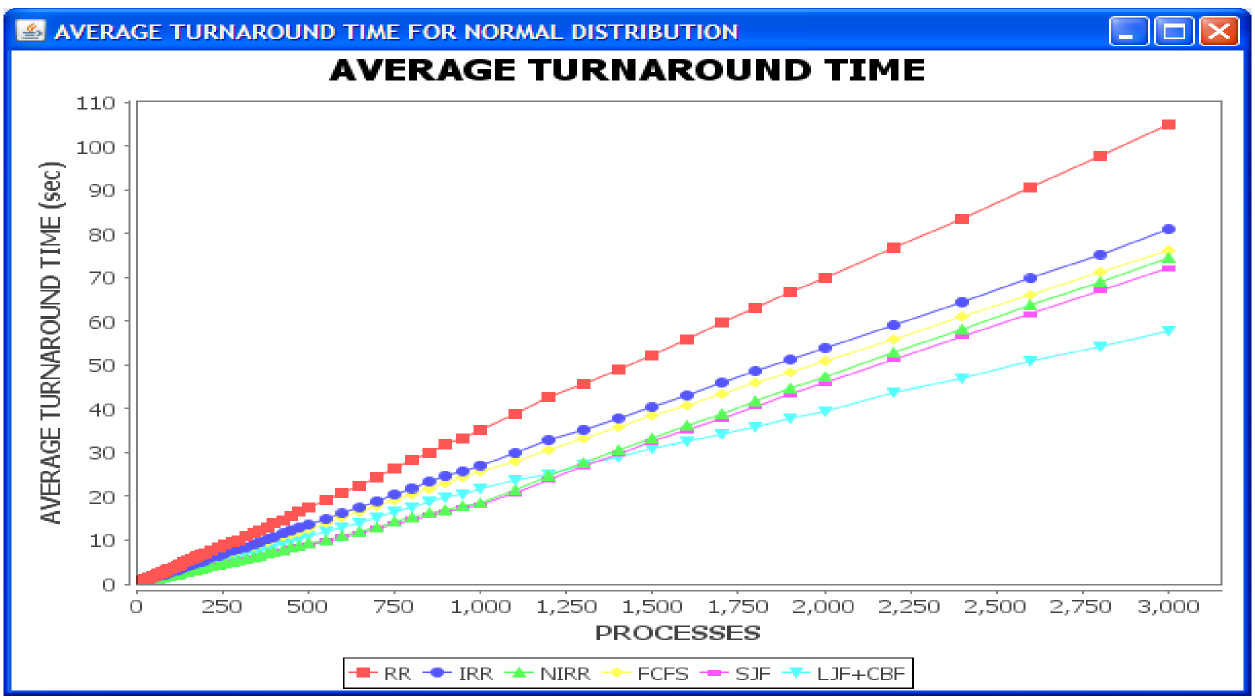
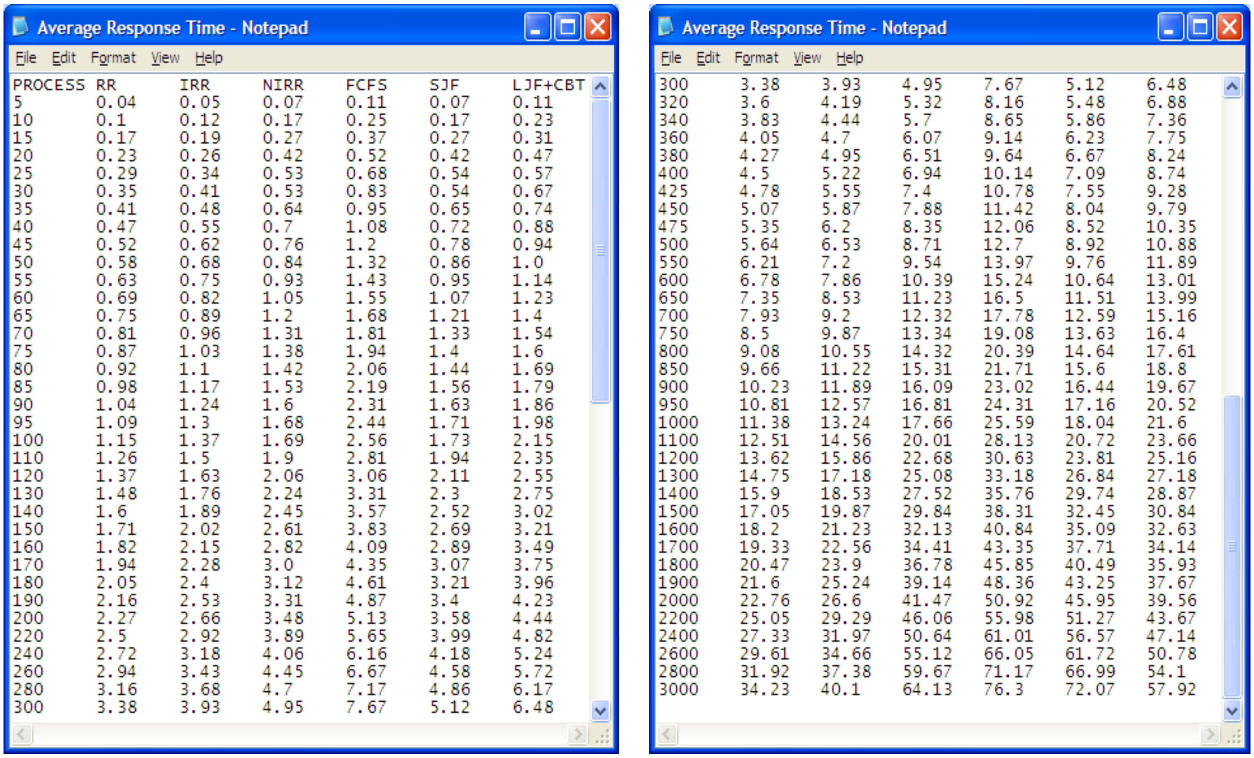


Figure 4.13: Graph of Average Turnaround Time using Normal Distribution



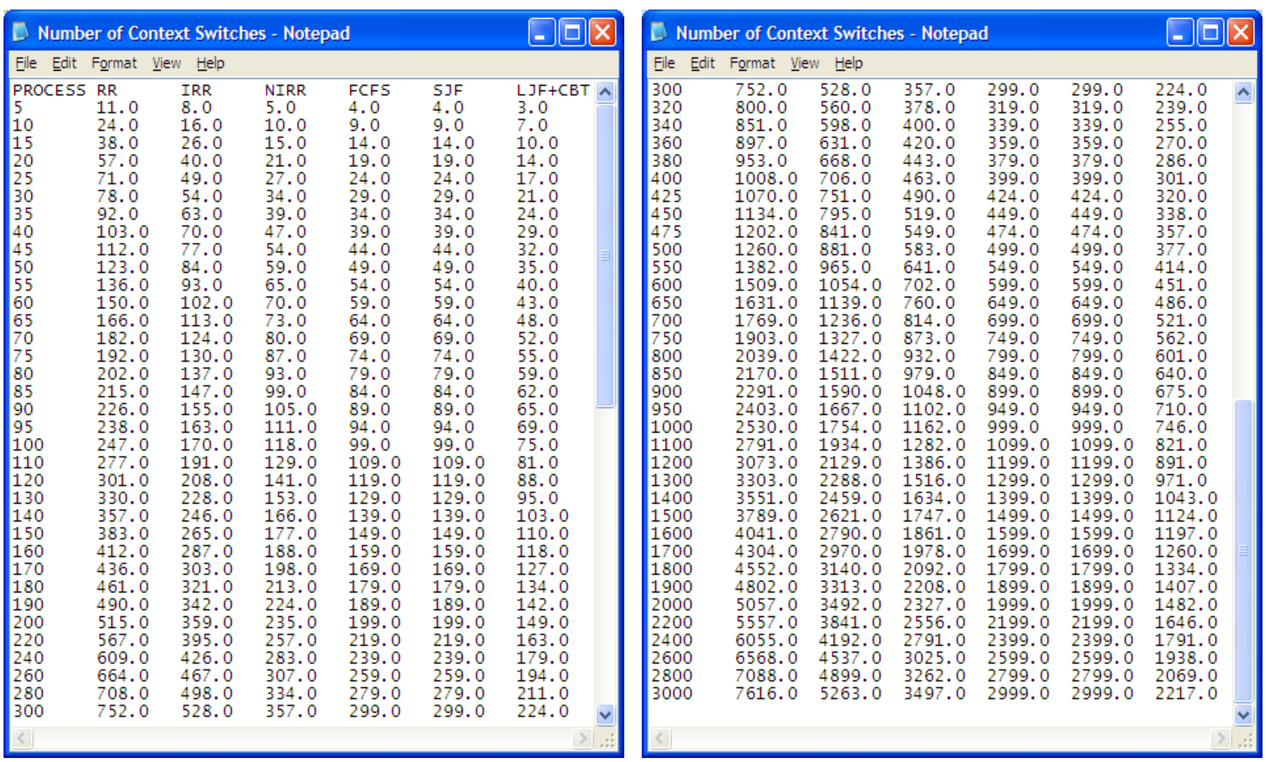
Average Response Time - Notepad

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.04	0.05	0.07	0.11	0.07	0.11
10	0.1	0.12	0.17	0.25	0.17	0.23
15	0.17	0.19	0.27	0.37	0.27	0.31
20	0.23	0.26	0.42	0.52	0.42	0.47
25	0.29	0.34	0.53	0.68	0.54	0.57
30	0.35	0.41	0.53	0.83	0.54	0.67
35	0.41	0.48	0.64	0.95	0.65	0.74
40	0.47	0.55	0.7	1.08	0.72	0.88
45	0.52	0.62	0.76	1.2	0.78	0.94
50	0.58	0.68	0.84	1.32	0.86	1.0
55	0.63	0.75	0.93	1.43	0.95	1.14
60	0.69	0.82	1.05	1.55	1.07	1.23
65	0.75	0.89	1.2	1.68	1.21	1.4
70	0.81	0.96	1.31	1.81	1.33	1.54
75	0.87	1.03	1.38	1.94	1.4	1.6
80	0.92	1.1	1.42	2.06	1.44	1.69
85	0.98	1.17	1.53	2.19	1.56	1.79
90	1.04	1.24	1.6	2.31	1.63	1.86
95	1.09	1.3	1.68	2.44	1.71	1.98
100	1.15	1.37	1.69	2.56	1.73	2.15
110	1.26	1.5	1.9	2.81	1.94	2.35
120	1.37	1.63	2.06	3.06	2.11	2.55
130	1.48	1.76	2.24	3.31	2.3	2.75
140	1.6	1.89	2.45	3.57	2.52	3.02
150	1.71	2.02	2.61	3.83	2.69	3.21
160	1.82	2.15	2.82	4.09	2.89	3.49
170	1.94	2.28	3.0	4.35	3.07	3.75
180	2.05	2.4	3.12	4.61	3.21	3.96
190	2.16	2.53	3.31	4.87	3.4	4.23
200	2.27	2.66	3.48	5.13	3.58	4.44
220	2.5	2.92	3.89	5.65	3.99	4.82
240	2.72	3.18	4.06	6.16	4.18	5.24
260	2.94	3.43	4.45	6.67	4.58	5.72
280	3.16	3.68	4.7	7.17	4.86	6.17
300	3.38	3.93	4.95	7.67	5.12	6.48

Average Response Time - Notepad

300	3.38	3.93	4.95	7.67	5.12	6.48
320	3.6	4.19	5.32	8.16	5.48	6.88
340	3.83	4.44	5.7	8.65	5.86	7.36
360	4.05	4.7	6.07	9.14	6.23	7.75
380	4.27	4.95	6.51	9.64	6.67	8.24
400	4.5	5.22	6.94	10.14	7.09	8.74
425	4.78	5.55	7.4	10.78	7.55	9.28
450	5.07	5.87	7.88	11.42	8.04	9.79
475	5.35	6.2	8.35	12.06	8.52	10.35
500	5.64	6.53	8.71	12.7	8.92	10.88
550	6.21	7.2	9.54	13.97	9.76	11.89
600	6.78	7.86	10.39	15.24	10.64	13.01
650	7.35	8.53	11.23	16.5	11.51	13.99
700	7.93	9.2	12.32	17.78	12.59	15.16
750	8.5	9.87	13.34	19.08	13.63	16.4
800	9.08	10.55	14.32	20.39	14.64	17.61
850	9.66	11.22	15.31	21.71	15.6	18.8
900	10.23	11.89	16.09	23.02	16.44	19.67
950	10.81	12.57	16.81	24.31	17.16	20.52
1000	11.38	13.24	17.66	25.59	18.04	21.6
1100	12.51	14.56	20.01	28.13	20.72	23.66
1200	13.62	15.86	22.68	30.63	23.81	25.16
1300	14.75	17.18	25.08	33.18	26.84	27.18
1400	15.9	18.53	27.52	35.76	29.74	28.87
1500	17.05	19.87	29.84	38.31	32.45	30.84
1600	18.2	21.23	32.13	40.84	35.09	32.63
1700	19.33	22.56	34.41	43.35	37.71	34.14
1800	20.47	23.9	36.78	45.85	40.49	35.93
1900	21.6	25.24	39.14	48.36	43.25	37.67
2000	22.76	26.6	41.47	50.92	45.95	39.56
2200	25.05	29.29	46.06	55.98	51.27	43.67
2400	27.33	31.97	50.64	61.01	56.57	47.14
2600	29.61	34.66	55.12	66.05	61.72	50.78
2800	31.92	37.38	59.67	71.17	66.99	54.1
3000	34.23	40.1	64.13	76.3	72.07	57.92

Figure 4.14: Results of Average Response Times obtained using Normal Distribution



Number of Context Switches - Notepad

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	11.0	8.0	5.0	4.0	4.0	3.0
10	24.0	16.0	10.0	9.0	9.0	7.0
15	38.0	26.0	15.0	14.0	14.0	10.0
20	57.0	40.0	21.0	19.0	19.0	14.0
25	71.0	49.0	27.0	24.0	24.0	17.0
30	78.0	54.0	34.0	29.0	29.0	21.0
35	92.0	63.0	39.0	34.0	34.0	24.0
40	103.0	70.0	47.0	39.0	39.0	29.0
45	112.0	77.0	54.0	44.0	44.0	32.0
50	123.0	84.0	59.0	49.0	49.0	35.0
55	136.0	93.0	65.0	54.0	54.0	40.0
60	150.0	102.0	70.0	59.0	59.0	43.0
65	166.0	113.0	73.0	64.0	64.0	48.0
70	182.0	124.0	80.0	69.0	69.0	52.0
75	192.0	130.0	87.0	74.0	74.0	55.0
80	202.0	137.0	93.0	79.0	79.0	59.0
85	215.0	147.0	99.0	84.0	84.0	62.0
90	226.0	155.0	105.0	89.0	89.0	65.0
95	238.0	163.0	111.0	94.0	94.0	69.0
100	247.0	170.0	118.0	99.0	99.0	75.0
110	277.0	191.0	129.0	109.0	109.0	81.0
120	301.0	208.0	141.0	119.0	119.0	88.0
130	330.0	228.0	153.0	129.0	129.0	95.0
140	357.0	246.0	166.0	139.0	139.0	103.0
150	383.0	265.0	177.0	149.0	149.0	110.0
160	412.0	287.0	188.0	159.0	159.0	118.0
170	436.0	303.0	198.0	169.0	169.0	127.0
180	461.0	321.0	213.0	179.0	179.0	134.0
190	490.0	342.0	224.0	189.0	189.0	142.0
200	515.0	359.0	235.0	199.0	199.0	149.0
220	567.0	395.0	257.0	219.0	219.0	163.0
240	609.0	426.0	283.0	239.0	239.0	179.0
260	664.0	467.0	307.0	259.0	259.0	194.0
280	708.0	498.0	334.0	279.0	279.0	211.0
300	752.0	528.0	357.0	299.0	299.0	224.0

Number of Context Switches - Notepad

300	752.0	528.0	357.0	299.0	299.0	224.0
320	800.0	560.0	378.0	319.0	319.0	239.0
340	851.0	598.0	400.0	339.0	339.0	255.0
360	897.0	631.0	420.0	359.0	359.0	270.0
380	953.0	668.0	443.0	379.0	379.0	286.0
400	1008.0	706.0	463.0	399.0	399.0	301.0
425	1070.0	751.0	490.0	424.0	424.0	320.0
450	1134.0	795.0	519.0	449.0	449.0	338.0
475	1202.0	841.0	549.0	474.0	474.0	357.0
500	1260.0	881.0	583.0	499.0	499.0	377.0
550	1382.0	965.0	641.0	549.0	549.0	414.0
600	1509.0	1054.0	702.0	599.0	599.0	451.0
650	1631.0	1139.0	760.0	649.0	649.0	486.0
700	1769.0	1236.0	814.0	699.0	699.0	521.0
750	1903.0	1327.0	873.0	749.0	749.0	562.0
800	2039.0	1422.0	932.0	799.0	799.0	601.0
850	2170.0	1511.0	979.0	849.0	849.0	640.0
900	2291.0	1590.0	1048.0	899.0	899.0	675.0
950	2403.0	1667.0	1102.0	949.0	949.0	710.0
1000	2530.0	1754.0	1162.0	999.0	999.0	746.0
1100	2791.0	1934.0	1282.0	1099.0	1099.0	821.0
1200	3073.0	2129.0	1386.0	1199.0	1199.0	891.0
1300	3303.0	2288.0	1516.0	1299.0	1299.0	971.0
1400	3551.0	2459.0	1634.0	1399.0	1399.0	1043.0
1500	3789.0	2621.0	1747.0	1499.0	1499.0	1124.0
1600	4041.0	2790.0	1861.0	1599.0	1599.0	1197.0
1700	4304.0	2970.0	1978.0	1699.0	1699.0	1260.0
1800	4552.0	3140.0	2092.0	1799.0	1799.0	1334.0
1900	4802.0	3313.0	2208.0	1899.0	1899.0	1407.0
2000	5057.0	3492.0	2327.0	1999.0	1999.0	1482.0
2200	5557.0	3841.0	2556.0	2199.0	2199.0	1646.0
2400	6055.0	4192.0	2791.0	2399.0	2399.0	1791.0
2600	6568.0	4537.0	3025.0	2599.0	2599.0	1938.0
2800	7088.0	4899.0	3262.0	2799.0	2799.0	2069.0
3000	7616.0	5263.0	3497.0	2999.0	2999.0	2217.0

Figure 4.15: Results of Number of Context Switches obtained using Normal Distribution

Figures 4.14 and 4.15 respectively present the results of Average Response Time and Number of Context Switches obtained for the different times of runs and different processes generated using normal distribution.

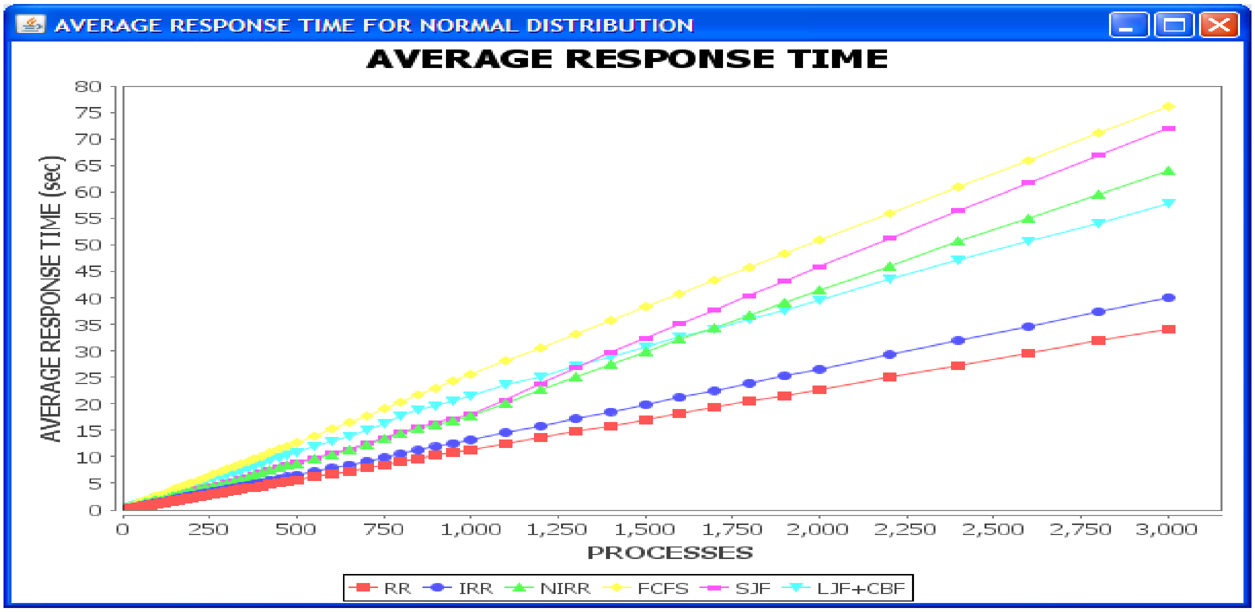


Figure 4.16: Graph of Average Response Time using Normal Distribution

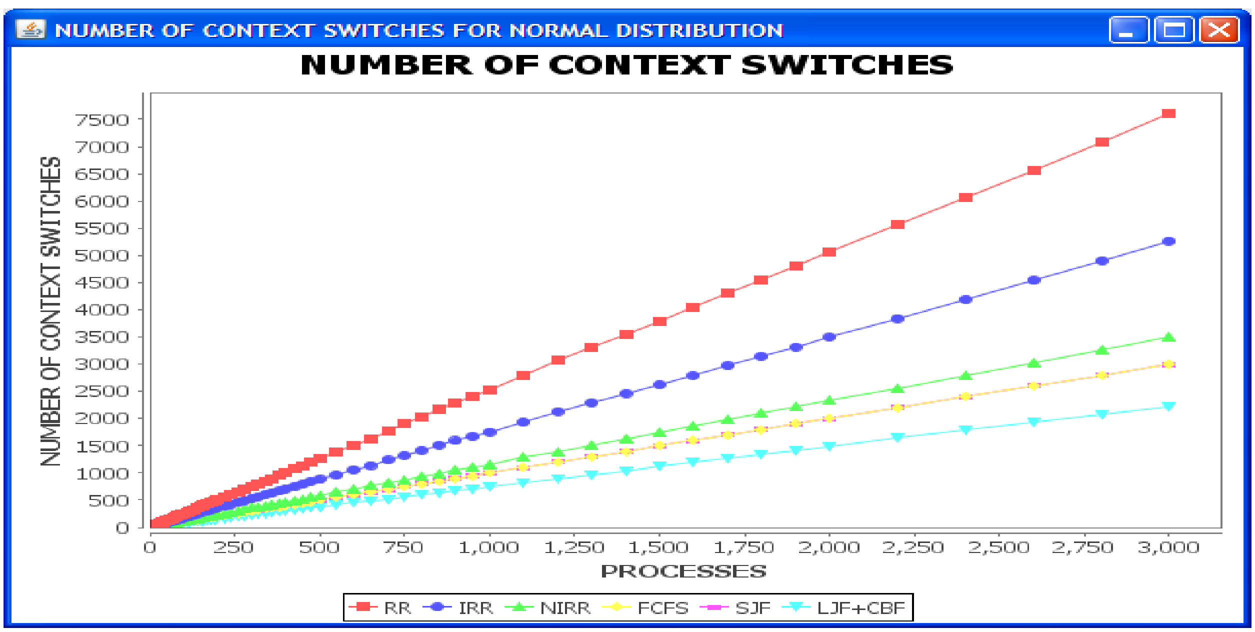


Figure 4.17: Graph of Number of Context Switches using Normal Distribution

Figure 4.16 shows the overall graphical result of the Average Response Time for all cases of values taken. It was observed from the graph that the RR has the best performance, IRR came second, the proposed algorithm (NIRR) came third followed by SJF, LJF+CBT and FCFS respectively when the number of processes is less than 1700. But when the number of processes exceeds 1700, RR and IRR still maintain the first and second positions, while LJF+CBT came third followed by the proposed algorithm (NIRR), SJF and FCFS respectively.

Figure 4.17 shows the overall graphical result of the Number of Context Switches for the same processes. It was observed from the graph that LJF+CBT produced the best performance, FCFS and SJF came second producing the same performance followed by the proposed algorithm (NIRR), IRR and RR respectively.

From the results of the five scheduling algorithms (FCFS, SJF, RR, IRR and LJF+CBT) compared with the proposed algorithm (NIRR), it has shown that the proposed algorithm (NIRR) has better Average Waiting Time (AWT) and Average Turnaround Time (ATAT) against LJF+CBT, FCFS, RR and IRR when the number of processes is less than 1500,but as the number of processes exceeds 1500, it has better Average Waiting Time (AWT) and Average Turnaround Time (ATAT) against FCFS, RR and IRR. It also has better Average Response Time (ART) against SJF, LJF+CBT and FCFS when the number of processes is less than 1700, but when number of processes exceeds 1700 it has better Average Response Time (ART) against SJF and FCFS, and better Number of Context Switches (NCS) against RR and IRR.

4.4.2 Results of uniform distribution

The following shows the relationships between the CPU scheduling algorithms under study using 3000 processes generated by uniform distribution with burst times ranging between 1 and 100*ms* and time quantum of 25*ms* used by RR and IRR.

Average Waiting Time - Notepad

File Edit Format View Help

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.16	0.13	0.12	0.13	0.09	0.11
10	0.32	0.25	0.22	0.26	0.16	0.24
15	0.43	0.34	0.27	0.37	0.21	0.29
20	0.62	0.47	0.39	0.49	0.32	0.39
25	0.85	0.64	0.52	0.62	0.45	0.57
30	0.93	0.72	0.55	0.75	0.46	0.63
35	1.12	0.87	0.67	0.88	0.57	0.77
40	1.24	0.97	0.72	1.0	0.62	0.82
45	1.38	1.09	0.81	1.12	0.7	0.97
50	1.52	1.19	0.87	1.23	0.77	1.03
55	1.75	1.36	1.0	1.35	0.89	1.17
60	1.9	1.49	1.09	1.47	0.98	1.3
65	2.06	1.61	1.17	1.59	1.05	1.36
70	2.3	1.79	1.32	1.72	1.19	1.54
75	2.52	1.97	1.47	1.85	1.32	1.72
80	2.69	2.1	1.55	1.99	1.4	1.79
85	2.86	2.23	1.63	2.12	1.48	1.9
90	2.98	2.33	1.71	2.26	1.54	1.95
95	3.14	2.46	1.79	2.38	1.62	2.11
100	3.28	2.59	1.85	2.51	1.69	2.22
110	3.72	2.93	2.09	2.77	1.93	2.45
120	4.08	3.21	2.28	3.03	2.11	2.63
130	4.48	3.51	2.5	3.29	2.31	2.89
140	4.69	3.71	2.65	3.54	2.41	3.08
150	4.97	3.94	2.82	3.8	2.56	3.31
160	5.2	4.12	2.92	4.05	2.65	3.5
170	5.59	4.44	3.17	4.3	2.86	3.76
180	5.9	4.68	3.35	4.55	3.02	3.96
190	6.29	4.98	3.58	4.8	3.24	4.21
200	6.66	5.27	3.77	5.06	3.42	4.45
220	7.47	5.93	4.2	5.58	3.86	4.99
240	8.12	6.47	4.57	6.11	4.19	5.44
260	8.76	7.0	4.94	6.63	4.52	5.89
280	9.65	7.67	5.42	7.16	4.99	6.44
300	10.37	8.25	5.83	7.7	5.37	6.89

Average Waiting Time - Notepad

File Edit Format View Help

300	10.37	8.25	5.83	7.7	5.37	6.89
320	11.02	8.76	6.15	8.23	5.67	7.27
340	11.58	9.23	6.5	8.75	5.95	7.75
360	12.25	9.78	6.87	9.27	6.3	8.17
380	12.79	10.24	7.15	9.79	6.56	8.61
400	13.38	10.72	7.45	10.31	6.85	8.98
425	14.04	11.27	7.91	10.94	7.18	9.53
450	14.94	11.97	8.41	11.56	7.64	10.09
475	15.76	12.61	8.87	12.19	8.06	10.65
500	16.69	13.34	9.31	12.82	8.56	11.15
550	18.23	14.57	10.23	14.08	9.33	12.28
600	19.57	15.69	10.92	15.32	9.96	13.24
650	20.84	16.71	11.57	16.53	10.56	14.11
700	22.54	18.09	12.6	17.73	11.48	15.35
750	24.24	19.39	13.57	18.95	12.36	16.42
800	25.57	20.5	14.28	20.17	13.01	17.41
850	27.21	21.79	15.22	21.38	13.85	18.52
900	29.08	23.27	16.34	22.61	14.88	19.82
950	30.69	24.53	17.27	23.85	15.7	20.94
1000	32.44	25.86	18.22	25.1	16.59	21.93
1100	35.52	28.31	19.91	27.57	18.15	24.13
1200	38.67	30.86	21.79	30.05	19.8	26.35
1300	42.0	33.47	23.63	32.54	21.52	28.52
1400	45.31	36.06	25.43	35.03	23.21	30.69
1500	48.74	38.8	27.4	37.54	25.0	32.99
1600	51.99	41.41	29.21	40.05	26.68	35.19
1700	55.6	44.23	31.31	42.58	28.59	37.62
1800	58.73	46.67	32.98	45.12	30.15	39.52
1900	61.58	49.02	34.59	47.64	31.54	41.6
2000	64.99	51.72	36.54	50.14	33.32	43.88
2200	72.25	57.39	40.72	55.2	37.12	48.67
2400	78.87	62.75	44.56	60.3	40.54	53.22
2600	85.2	67.83	48.06	65.39	43.73	57.47
2800	92.04	73.17	52.0	70.45	47.26	61.94
3000	98.51	78.38	55.61	75.53	50.5	66.43

Figure 4.18: Results of Average Waiting Times obtained using Uniform Distribution

Average Turnaround Time - Notepad

File Edit Format View Help

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.22	0.19	0.18	0.19	0.15	0.17
10	0.38	0.31	0.27	0.32	0.22	0.29
15	0.48	0.39	0.32	0.42	0.26	0.34
20	0.67	0.52	0.44	0.54	0.37	0.44
25	0.91	0.7	0.58	0.67	0.5	0.63
30	0.98	0.77	0.6	0.8	0.51	0.68
35	1.17	0.92	0.72	0.93	0.62	0.82
40	1.29	1.02	0.77	1.05	0.67	0.87
45	1.43	1.13	0.85	1.17	0.75	1.02
50	1.57	1.24	0.92	1.28	0.82	1.08
55	1.8	1.41	1.05	1.4	0.94	1.22
60	1.95	1.54	1.14	1.52	1.03	1.35
65	2.11	1.66	1.22	1.64	1.1	1.41
70	2.35	1.84	1.38	1.77	1.24	1.6
75	2.57	2.02	1.52	1.91	1.37	1.77
80	2.74	2.15	1.61	2.04	1.45	1.84
85	2.91	2.28	1.68	2.18	1.53	1.95
90	3.04	2.38	1.76	2.31	1.59	2.0
95	3.19	2.51	1.84	2.43	1.67	2.16
100	3.33	2.64	1.9	2.56	1.74	2.27
110	3.77	2.99	2.15	2.82	1.98	2.5
120	4.14	3.26	2.33	3.08	2.16	2.68
130	4.53	3.56	2.55	3.34	2.36	2.94
140	4.74	3.76	2.7	3.6	2.46	3.13
150	5.03	3.99	2.87	3.85	2.61	3.37
160	5.25	4.17	2.97	4.1	2.7	3.55
170	5.64	4.49	3.22	4.35	2.91	3.81
180	5.95	4.73	3.4	4.6	3.07	4.01
190	6.34	5.03	3.63	4.85	3.29	4.26
200	6.71	5.32	3.82	5.11	3.47	4.5
220	7.52	5.98	4.25	5.63	3.91	5.05
240	8.17	6.53	4.63	6.16	4.25	5.49
260	8.81	7.05	4.99	6.68	4.57	5.94
280	9.7	7.73	5.47	7.21	5.05	6.49
300	10.43	8.31	5.88	7.75	5.42	6.94

Average Turnaround Time - Notepad

File Edit Format View Help

300	10.43	8.31	5.88	7.75	5.42	6.94
320	11.07	8.81	6.21	8.28	5.72	7.32
340	11.63	9.28	6.55	8.8	6.0	7.8
360	12.3	9.83	6.92	9.33	6.35	8.23
380	12.84	10.29	7.2	9.85	6.61	8.66
400	13.43	10.78	7.51	10.36	6.9	9.03
425	14.09	11.32	7.96	10.99	7.23	9.58
450	14.99	12.02	8.46	11.61	7.69	10.14
475	15.81	12.66	8.93	12.24	8.11	10.7
500	16.74	13.39	9.36	12.87	8.61	11.2
550	18.28	14.62	10.28	14.13	9.38	12.33
600	19.62	15.74	10.97	15.38	10.01	13.29
650	20.89	16.76	11.62	16.58	10.61	14.16
700	22.59	18.14	12.65	17.78	11.53	15.4
750	24.29	19.44	13.62	19.0	12.41	16.47
800	25.62	20.54	14.32	20.22	13.06	17.45
850	27.26	21.84	15.27	21.43	13.9	18.57
900	29.13	23.32	16.39	22.66	14.93	19.87
950	30.74	24.58	17.32	23.9	15.75	20.99
1000	32.49	25.91	18.27	25.15	16.64	21.98
1100	35.57	28.36	19.96	27.62	18.2	24.18
1200	38.72	30.91	21.84	30.1	19.85	26.4
1300	42.05	33.52	23.68	32.59	21.57	28.57
1400	45.36	36.11	25.48	35.08	23.26	30.74
1500	48.79	38.85	27.45	37.59	25.05	33.04
1600	52.04	41.46	29.26	40.1	26.73	35.24
1700	55.65	44.28	31.36	42.63	28.64	37.67
1800	58.78	46.72	33.03	45.17	30.2	39.58
1900	61.63	49.07	34.64	47.69	31.59	41.65
2000	65.04	51.77	36.59	50.19	33.37	43.93
2200	72.3	57.44	40.77	55.25	37.17	48.72
2400	78.92	62.8	44.61	60.35	40.59	53.27
2600	85.25	67.88	48.11	65.44	43.78	57.52
2800	92.09	73.22	52.05	70.5	47.31	61.99
3000	98.56	78.43	55.66	75.58	50.56	66.48

Figure 4.19: Results of Average Turnaround Times obtained using Uniform Distribution

Figures 4.18 and 4.19 respectively present the results of Average Waiting Time and Average Turnaround Time obtained for the different times of runs and different processes generated using uniform distribution.

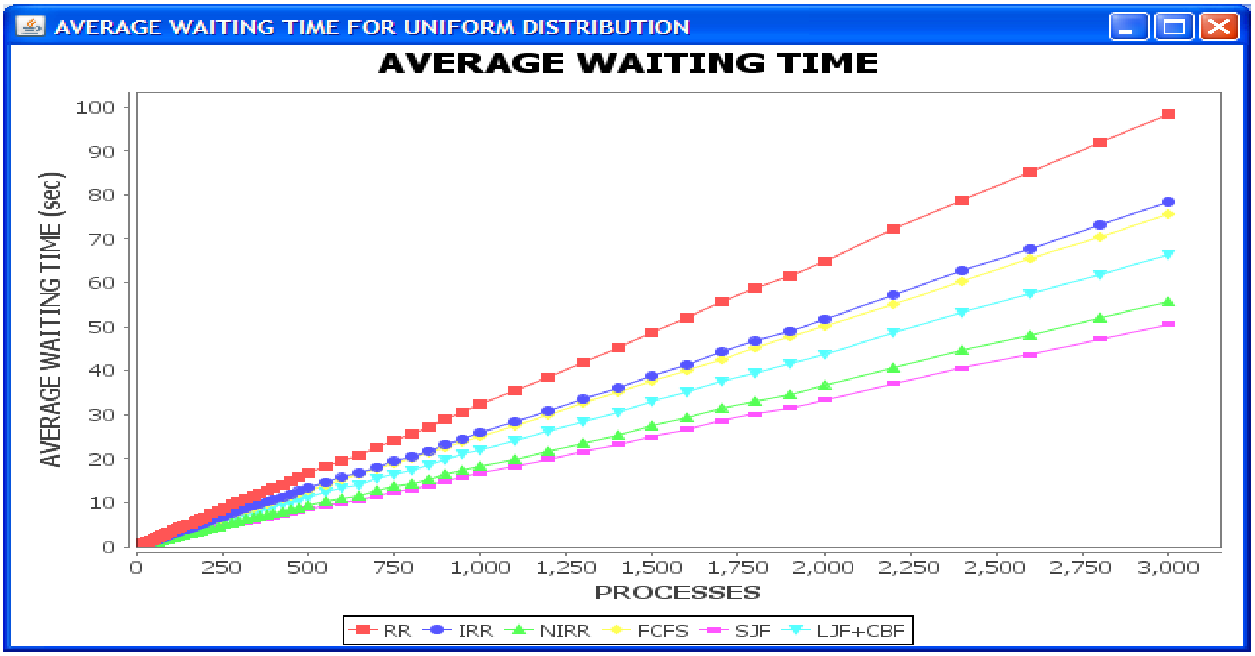


Figure 4.20: Graph of Average Waiting Time using Uniform Distribution

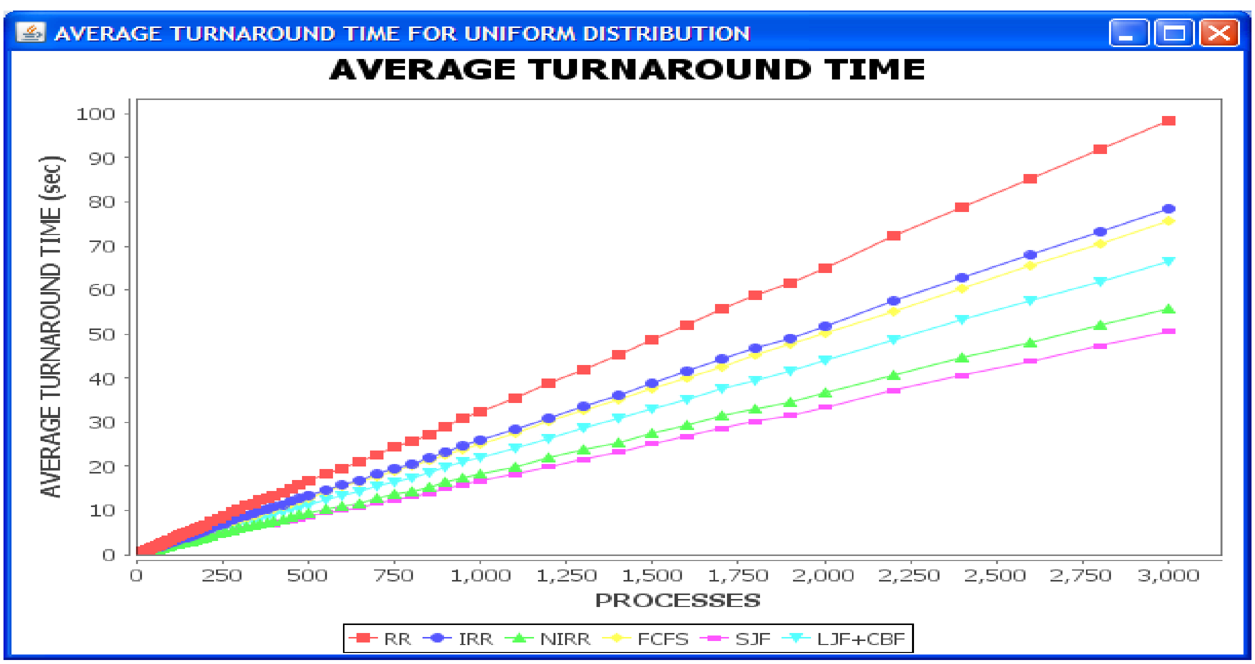
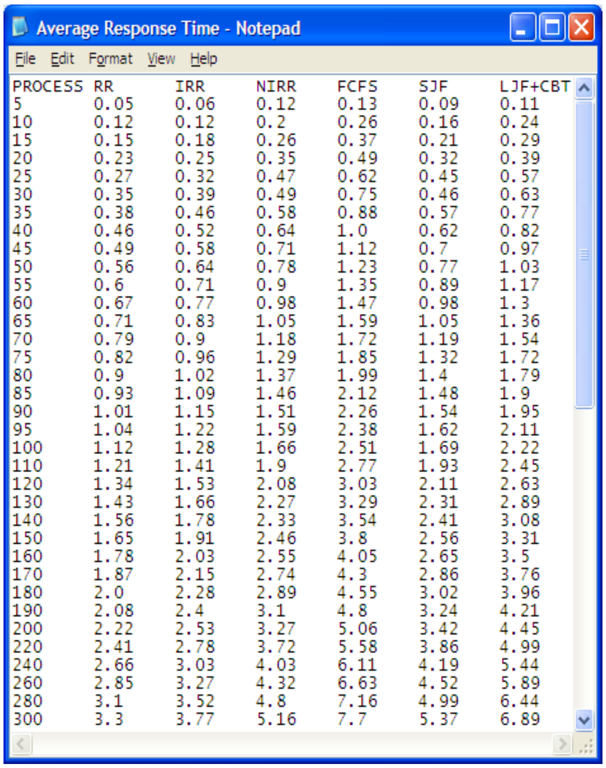


Figure 4.21: Graph of Average Turnaround Time using Uniform Distribution

Figure 4.20 shows the overall graphical result of the Average Waiting Time for all cases of values taken. It was observed from the graph that the SJF produced the best performance, and then the proposed algorithm (NIRR) followed by LJF+CBT, FCFS IRR and RR respectively.

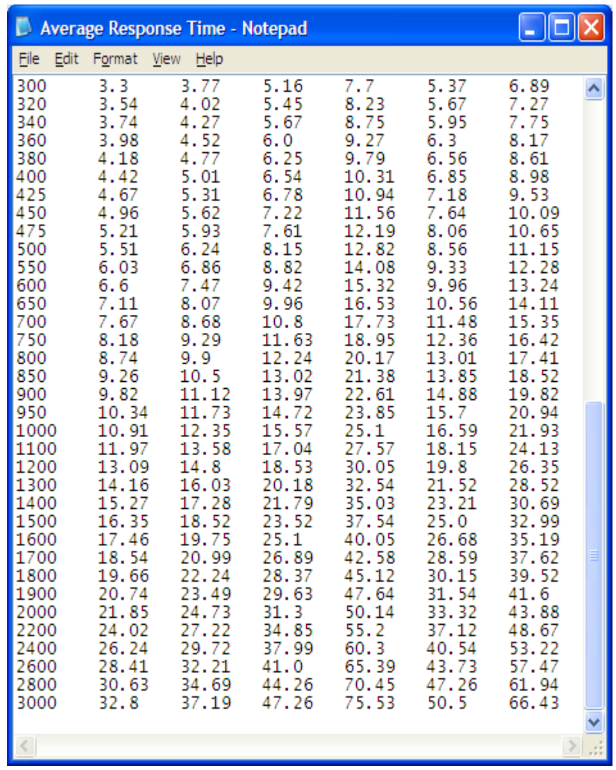
Figure 4.21 shows the overall graphical result of the Average Turnaround Time for all cases of values taken. It was observed from the graph that the SJF produced the best performance, and then the proposed algorithm (NIRR) followed by LJF+CBT, FCFS IRR and RR respectively.



Average Response Time - Notepad

File Edit Format View Help

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.05	0.06	0.12	0.13	0.09	0.11
10	0.12	0.12	0.2	0.26	0.16	0.24
15	0.15	0.18	0.26	0.37	0.21	0.29
20	0.23	0.25	0.35	0.49	0.32	0.39
25	0.27	0.32	0.47	0.62	0.45	0.57
30	0.35	0.39	0.49	0.75	0.46	0.63
35	0.38	0.46	0.58	0.88	0.57	0.77
40	0.46	0.52	0.64	1.0	0.62	0.82
45	0.49	0.58	0.71	1.12	0.7	0.97
50	0.56	0.64	0.78	1.23	0.77	1.03
55	0.6	0.71	0.9	1.35	0.89	1.17
60	0.67	0.77	0.98	1.47	0.98	1.3
65	0.71	0.83	1.05	1.59	1.05	1.36
70	0.79	0.9	1.18	1.72	1.19	1.54
75	0.82	0.96	1.29	1.85	1.32	1.72
80	0.9	1.02	1.37	1.99	1.4	1.79
85	0.93	1.09	1.46	2.12	1.48	1.9
90	1.01	1.15	1.51	2.26	1.54	1.95
95	1.04	1.22	1.59	2.38	1.62	2.11
100	1.12	1.28	1.66	2.51	1.69	2.22
110	1.21	1.41	1.9	2.77	1.93	2.45
120	1.34	1.53	2.08	3.03	2.11	2.63
130	1.43	1.66	2.27	3.29	2.31	2.89
140	1.56	1.78	2.33	3.54	2.41	3.08
150	1.65	1.91	2.46	3.8	2.56	3.31
160	1.78	2.03	2.55	4.05	2.65	3.5
170	1.87	2.15	2.74	4.3	2.86	3.76
180	2.0	2.28	2.89	4.55	3.02	3.96
190	2.08	2.4	3.1	4.8	3.24	4.21
200	2.22	2.53	3.27	5.06	3.42	4.45
220	2.41	2.78	3.72	5.58	3.86	4.99
240	2.66	3.03	4.03	6.11	4.19	5.44
260	2.85	3.27	4.32	6.63	4.52	5.89
280	3.1	3.52	4.8	7.16	4.99	6.44
300	3.3	3.77	5.16	7.7	5.37	6.89



Average Response Time - Notepad

File Edit Format View Help

300	3.3	3.77	5.16	7.7	5.37	6.89
320	3.54	4.02	5.45	8.23	5.67	7.27
340	3.74	4.27	5.67	8.75	5.95	7.75
360	3.98	4.52	6.0	9.27	6.3	8.17
380	4.18	4.77	6.25	9.79	6.56	8.61
400	4.42	5.01	6.54	10.31	6.85	8.98
425	4.67	5.31	6.78	10.94	7.18	9.53
450	4.96	5.62	7.22	11.56	7.64	10.09
475	5.21	5.93	7.61	12.19	8.06	10.65
500	5.51	6.24	8.15	12.82	8.56	11.15
550	6.03	6.86	8.82	14.08	9.33	12.28
600	6.6	7.47	9.42	15.32	9.96	13.24
650	7.11	8.07	9.96	16.53	10.56	14.11
700	7.67	8.68	10.8	17.73	11.48	15.35
750	8.18	9.29	11.63	18.95	12.36	16.42
800	8.74	9.9	12.24	20.17	13.01	17.41
850	9.26	10.5	13.02	21.38	13.85	18.52
900	9.82	11.12	13.97	22.61	14.88	19.82
950	10.34	11.73	14.72	23.85	15.7	20.94
1000	10.91	12.35	15.57	25.1	16.59	21.93
1100	11.97	13.58	17.04	27.57	18.15	24.13
1200	13.09	14.8	18.53	30.05	19.8	26.35
1300	14.16	16.03	20.18	32.54	21.52	28.52
1400	15.27	17.28	21.79	35.03	23.21	30.69
1500	16.35	18.52	23.52	37.54	25.0	32.99
1600	17.46	19.75	25.1	40.05	26.68	35.19
1700	18.54	20.99	26.89	42.58	28.59	37.62
1800	19.66	22.24	28.37	45.12	30.15	39.52
1900	20.74	23.49	29.63	47.64	31.54	41.6
2000	21.85	24.73	31.3	50.14	33.32	43.88
2200	24.02	27.22	34.85	55.2	37.12	48.67
2400	26.24	29.72	37.99	60.3	40.54	53.22
2600	28.41	32.21	41.0	65.39	43.73	57.47
2800	30.63	34.69	44.26	70.45	47.26	61.94
3000	32.8	37.19	47.26	75.53	50.5	66.43

Figure 4.22: Results of Average Response Times obtained using Uniform Distribution

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	13.0	9.0	5.0	4.0	4.0	3.0
10	26.0	18.0	12.0	9.0	9.0	7.0
15	35.0	24.0	17.0	14.0	14.0	10.0
20	49.0	33.0	24.0	19.0	19.0	13.0
25	65.0	44.0	30.0	24.0	24.0	18.0
30	73.0	50.0	36.0	29.0	29.0	21.0
35	87.0	60.0	43.0	34.0	34.0	25.0
40	97.0	67.0	48.0	39.0	39.0	28.0
45	108.0	75.0	55.0	44.0	44.0	33.0
50	119.0	82.0	60.0	49.0	49.0	36.0
55	135.0	93.0	66.0	54.0	54.0	40.0
60	147.0	102.0	72.0	59.0	59.0	44.0
65	159.0	110.0	78.0	64.0	64.0	47.0
70	175.0	121.0	85.0	69.0	69.0	52.0
75	192.0	134.0	92.0	74.0	74.0	56.0
80	204.0	142.0	98.0	79.0	79.0	59.0
85	215.0	149.0	103.0	84.0	84.0	63.0
90	225.0	156.0	110.0	89.0	89.0	66.0
95	237.0	165.0	116.0	94.0	94.0	71.0
100	248.0	173.0	121.0	99.0	99.0	75.0
110	278.0	195.0	132.0	109.0	109.0	82.0
120	303.0	212.0	143.0	119.0	119.0	89.0
130	331.0	231.0	156.0	129.0	129.0	97.0
140	352.0	248.0	172.0	139.0	139.0	104.0
150	375.0	264.0	185.0	149.0	149.0	112.0
160	396.0	279.0	197.0	159.0	159.0	119.0
170	425.0	300.0	211.0	169.0	169.0	127.0
180	449.0	317.0	224.0	179.0	179.0	134.0
190	475.0	335.0	236.0	189.0	189.0	142.0
200	502.0	354.0	248.0	199.0	199.0	150.0
220	561.0	398.0	269.0	219.0	219.0	166.0
240	611.0	436.0	295.0	239.0	239.0	181.0
260	661.0	472.0	321.0	259.0	259.0	196.0
280	720.0	512.0	343.0	279.0	279.0	212.0
300	772.0	549.0	368.0	299.0	299.0	227.0

300	772.0	549.0	368.0	299.0	299.0	227.0
320	821.0	583.0	392.0	319.0	319.0	241.0
340	867.0	617.0	421.0	339.0	339.0	257.0
360	917.0	653.0	445.0	359.0	359.0	272.0
380	962.0	686.0	469.0	379.0	379.0	287.0
400	1008.0	719.0	492.0	399.0	399.0	301.0
425	1063.0	759.0	531.0	424.0	424.0	321.0
450	1129.0	805.0	562.0	449.0	449.0	340.0
475	1192.0	849.0	594.0	474.0	474.0	359.0
500	1260.0	897.0	616.0	499.0	499.0	375.0
550	1376.0	978.0	685.0	549.0	549.0	415.0
600	1491.0	1062.0	746.0	599.0	599.0	451.0
650	1599.0	1139.0	808.0	649.0	649.0	486.0
700	1729.0	1234.0	873.0	699.0	699.0	526.0
750	1856.0	1322.0	936.0	749.0	749.0	563.0
800	1969.0	1404.0	997.0	799.0	799.0	599.0
850	2096.0	1494.0	1061.0	849.0	849.0	637.0
900	2231.0	1590.0	1125.0	899.0	899.0	677.0
950	2357.0	1679.0	1190.0	949.0	949.0	714.0
1000	2485.0	1766.0	1251.0	999.0	999.0	749.0
1100	2726.0	1936.0	1374.0	1099.0	1099.0	825.0
1200	2975.0	2117.0	1505.0	1199.0	1199.0	900.0
1300	3222.0	2289.0	1627.0	1299.0	1299.0	975.0
1400	3470.0	2463.0	1748.0	1399.0	1399.0	1050.0
1500	3734.0	2653.0	1871.0	1499.0	1499.0	1124.0
1600	3980.0	2831.0	1994.0	1599.0	1599.0	1199.0
1700	4244.0	3016.0	2122.0	1699.0	1699.0	1277.0
1800	4482.0	3180.0	2243.0	1799.0	1799.0	1347.0
1900	4720.0	3355.0	2372.0	1899.0	1899.0	1419.0
2000	4975.0	3537.0	2498.0	1999.0	1999.0	1495.0
2200	5506.0	3909.0	2753.0	2199.0	2199.0	1651.0
2400	6012.0	4276.0	3011.0	2399.0	2399.0	1802.0
2600	6502.0	4625.0	3260.0	2599.0	2599.0	1950.0
2800	7015.0	4985.0	3517.0	2799.0	2799.0	2099.0
3000	7520.0	5348.0	3771.0	2999.0	2999.0	2250.0

Figure 4.23: Results of Number of Context Switches obtained using Uniform Distribution

Figures 4.22 and 4.23 respectively present the results of Average Response Time and Number of Context Switches obtained for the different times of runs and different processes generated using uniform distribution.

Figure 4.24 shows the overall graphical result of the Average Response Time for all cases of values taken. It was observed from the graph that RR produces the best performance, IRR came second, the proposed algorithm (NIRR) came third followed by SJF, LJF+CBT and FCFS respectively. Figure 4.25 shows the overall graphical result of the Number of Context Switches for the same processes. It was observed from the graph that LJF+CBT produced the best performance, FCFS and SJF came second producing the same performance followed by the proposed algorithm (NIRR), IRR and RR respectively.

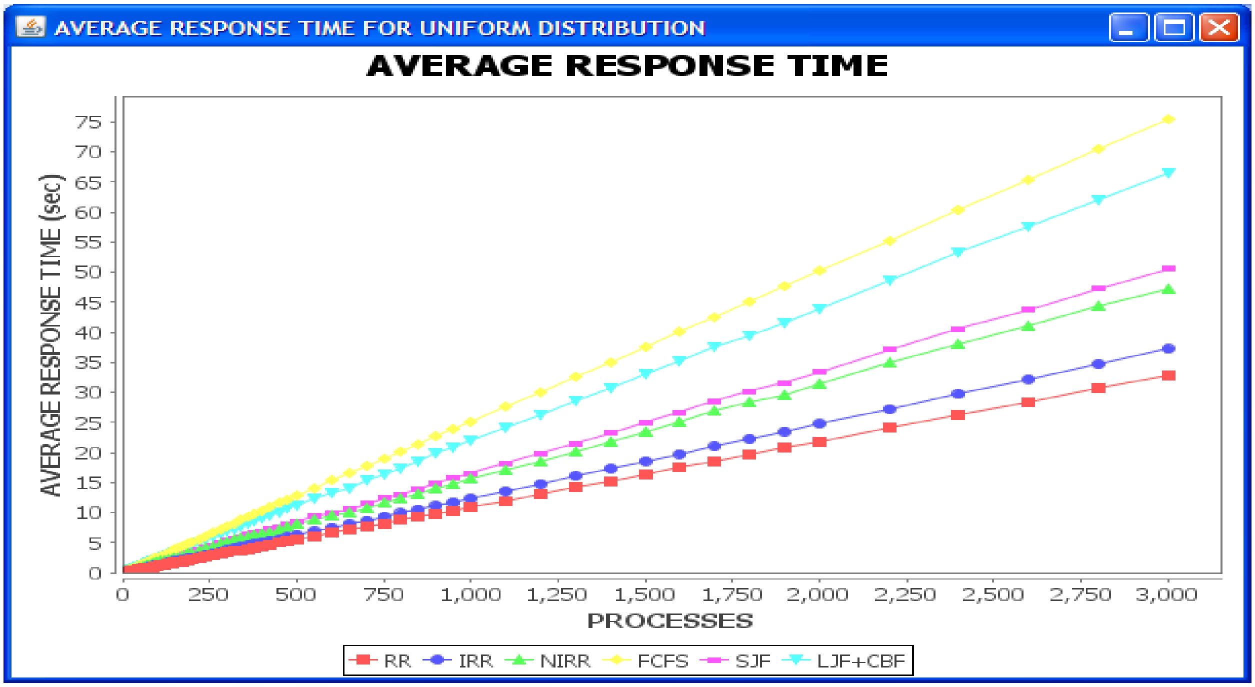


Figure 4.24: Graph of Average Response Time using Uniform Distribution

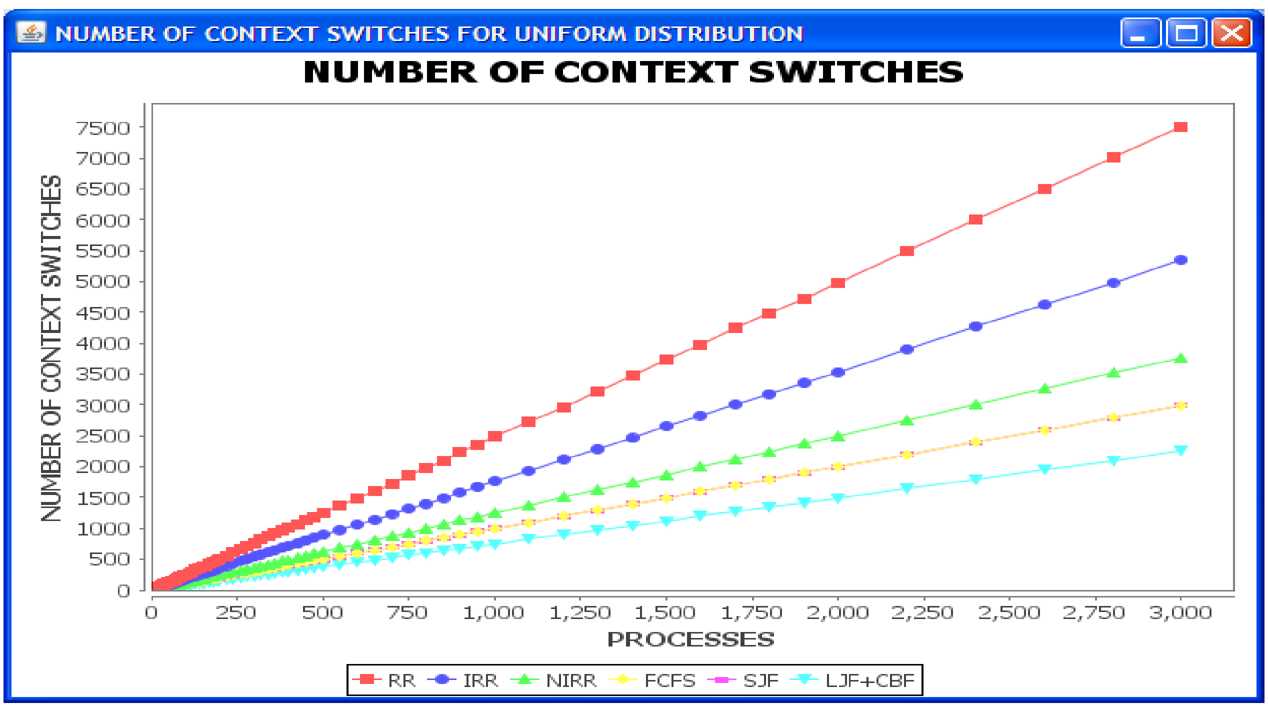


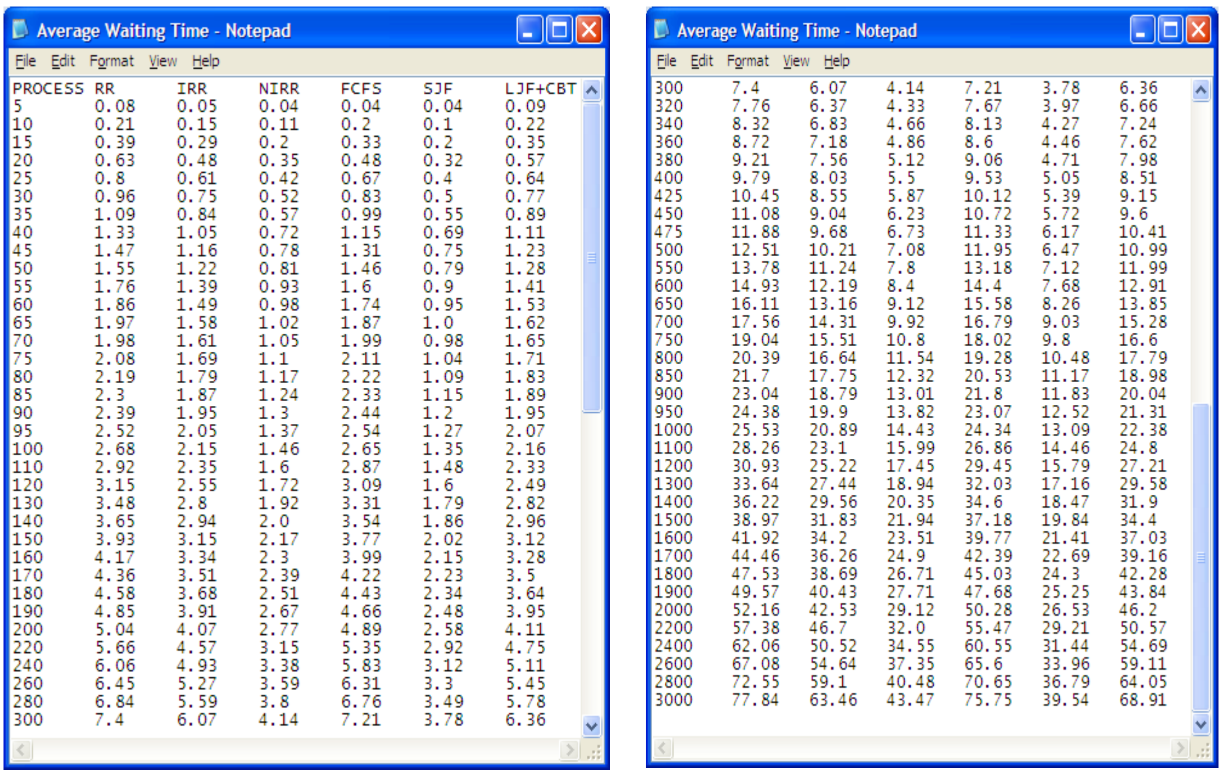
Figure 4.25: Graph of Number of Context Switches using Uniform Distribution

From the results of the five scheduling algorithms (FCFS, SJF, RR, IRR and LJF+CBT) compared with the proposed algorithm (NIRR), it has shown that NIRR produced better Average

Waiting Time (AWT) and Average Turnaround Time (ATAT) against LJF+CBT ,FCFS, IRR and RR respectively. It also produced better Average Response Time (ART) against SJF, LJF+CBT and FCFS, and better Number of Context Switches (NCS) against RR and IRR.

4.4.3 Results of exponential distribution

The following shows the relationships between the CPU scheduling algorithms under study using 3000 processes generated by exponential distribution with a rate (λ) of 0.02 *per ms*.



PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.08	0.05	0.04	0.04	0.04	0.09
10	0.21	0.15	0.11	0.2	0.1	0.22
15	0.39	0.29	0.2	0.33	0.2	0.35
20	0.63	0.48	0.35	0.48	0.32	0.57
25	0.8	0.61	0.42	0.67	0.4	0.64
30	0.96	0.75	0.52	0.83	0.5	0.77
35	1.09	0.84	0.57	0.99	0.55	0.89
40	1.33	1.05	0.72	1.15	0.69	1.11
45	1.47	1.16	0.78	1.31	0.75	1.23
50	1.55	1.22	0.81	1.46	0.79	1.28
55	1.76	1.39	0.93	1.6	0.9	1.41
60	1.86	1.49	0.98	1.74	0.95	1.53
65	1.97	1.58	1.02	1.87	1.0	1.62
70	1.98	1.61	1.05	1.99	0.98	1.65
75	2.08	1.69	1.1	2.11	1.04	1.71
80	2.19	1.79	1.17	2.22	1.09	1.83
85	2.3	1.87	1.24	2.33	1.15	1.89
90	2.39	1.95	1.3	2.44	1.2	1.95
95	2.52	2.05	1.37	2.54	1.27	2.07
100	2.68	2.15	1.46	2.65	1.35	2.16
110	2.92	2.35	1.6	2.87	1.48	2.33
120	3.15	2.55	1.72	3.09	1.6	2.49
130	3.48	2.8	1.92	3.31	1.79	2.82
140	3.65	2.94	2.0	3.54	1.86	2.96
150	3.93	3.15	2.17	3.77	2.02	3.12
160	4.17	3.34	2.3	3.99	2.15	3.28
170	4.36	3.51	2.39	4.22	2.23	3.5
180	4.58	3.68	2.51	4.43	2.34	3.64
190	4.85	3.91	2.67	4.66	2.48	3.95
200	5.04	4.07	2.77	4.89	2.58	4.11
220	5.66	4.57	3.15	5.35	2.92	4.75
240	6.06	4.93	3.38	5.83	3.12	5.11
260	6.45	5.27	3.59	6.31	3.3	5.45
280	6.84	5.59	3.8	6.76	3.49	5.78
300	7.4	6.07	4.14	7.21	3.78	6.36

300	7.4	6.07	4.14	7.21	3.78	6.36
320	7.76	6.37	4.33	7.67	3.97	6.66
340	8.32	6.83	4.66	8.13	4.27	7.24
360	8.72	7.18	4.86	8.6	4.46	7.62
380	9.21	7.56	5.12	9.06	4.71	7.98
400	9.79	8.03	5.5	9.53	5.05	8.51
425	10.45	8.55	5.87	10.12	5.39	9.15
450	11.08	9.04	6.23	10.72	5.72	9.6
475	11.88	9.68	6.73	11.33	6.17	10.41
500	12.51	10.21	7.08	11.95	6.47	10.99
550	13.78	11.24	7.8	13.18	7.12	11.99
600	14.93	12.19	8.4	14.4	7.68	12.91
650	16.11	13.16	9.12	15.58	8.26	13.85
700	17.56	14.31	9.92	16.79	9.03	15.28
750	19.04	15.51	10.8	18.02	9.8	16.6
800	20.39	16.64	11.54	19.28	10.48	17.79
850	21.7	17.75	12.32	20.53	11.17	18.98
900	23.04	18.79	13.01	21.8	11.83	20.04
950	24.38	19.9	13.82	23.07	12.52	21.31
1000	25.53	20.89	14.43	24.34	13.09	22.38
1100	28.26	23.1	15.99	26.86	14.46	24.8
1200	30.93	25.22	17.45	29.45	15.79	27.21
1300	33.64	27.44	18.94	32.03	17.16	29.58
1400	36.22	29.56	20.35	34.6	18.47	31.9
1500	38.97	31.83	21.94	37.18	19.84	34.4
1600	41.92	34.2	23.51	39.77	21.41	37.03
1700	44.46	36.26	24.9	42.39	22.69	39.16
1800	47.53	38.69	26.71	45.03	24.3	42.28
1900	49.57	40.43	27.71	47.68	25.25	43.84
2000	52.16	42.53	29.12	50.28	26.53	46.2
2200	57.38	46.7	32.0	55.47	29.21	50.57
2400	62.06	50.52	34.55	60.55	31.44	54.69
2600	67.08	54.64	37.35	65.6	33.96	59.11
2800	72.55	59.1	40.48	70.65	36.79	64.05
3000	77.84	63.46	43.47	75.75	39.54	68.91

Figure 4.26: Results of Average Waiting Times obtained using Exponential Distribution

Average Turnaround Time - Notepad

File Edit Format View Help

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.13	0.1	0.1	0.1	0.1	0.14
10	0.26	0.2	0.16	0.25	0.15	0.27
15	0.44	0.34	0.25	0.38	0.25	0.41
20	0.7	0.54	0.41	0.54	0.39	0.63
25	0.86	0.68	0.48	0.73	0.47	0.7
30	1.02	0.82	0.58	0.89	0.56	0.83
35	1.15	0.9	0.63	1.04	0.61	0.94
40	1.39	1.11	0.78	1.21	0.75	1.17
45	1.53	1.22	0.84	1.37	0.81	1.29
50	1.61	1.28	0.87	1.51	0.85	1.34
55	1.82	1.45	0.98	1.66	0.96	1.47
60	1.92	1.54	1.03	1.79	1.01	1.59
65	2.03	1.63	1.08	1.93	1.05	1.67
70	2.03	1.66	1.1	2.05	1.03	1.71
75	2.14	1.74	1.16	2.16	1.09	1.76
80	2.24	1.84	1.22	2.27	1.14	1.88
85	2.35	1.92	1.29	2.38	1.2	1.94
90	2.44	2.0	1.35	2.49	1.25	2.0
95	2.57	2.1	1.42	2.59	1.32	2.11
100	2.73	2.2	1.51	2.7	1.4	2.21
110	2.97	2.4	1.65	2.92	1.53	2.37
120	3.2	2.6	1.76	3.13	1.65	2.54
130	3.53	2.85	1.97	3.36	1.84	2.86
140	3.7	2.99	2.05	3.59	1.91	3.01
150	3.97	3.2	2.22	3.82	2.07	3.17
160	4.22	3.38	2.34	4.04	2.2	3.33
170	4.41	3.55	2.43	4.26	2.28	3.54
180	4.63	3.73	2.55	4.48	2.39	3.69
190	4.9	3.96	2.72	4.71	2.53	3.99
200	5.09	4.12	2.82	4.93	2.63	4.16
220	5.71	4.61	3.2	5.4	2.97	4.8
240	6.11	4.97	3.43	5.88	3.16	5.16
260	6.5	5.32	3.64	6.35	3.35	5.5
280	6.89	5.64	3.85	6.8	3.54	5.83
300	7.45	6.11	4.19	7.26	3.83	6.41

Average Turnaround Time - Notepad

File Edit Format View Help

300	7.45	6.11	4.19	7.26	3.83	6.41
320	7.8	6.42	4.38	7.72	4.01	6.7
340	8.37	6.88	4.71	8.18	4.32	7.29
360	8.77	7.23	4.9	8.65	4.51	7.67
380	9.25	7.61	5.17	9.11	4.76	8.03
400	9.84	8.08	5.55	9.57	5.1	8.55
425	10.49	8.6	5.92	10.17	5.43	9.2
450	11.13	9.09	6.27	10.77	5.77	9.65
475	11.93	9.73	6.78	11.38	6.22	10.46
500	12.56	10.26	7.13	12.0	6.52	11.04
550	13.83	11.29	7.84	13.23	7.17	12.04
600	14.98	12.24	8.45	14.44	7.73	12.96
650	16.16	13.21	9.17	15.63	8.31	13.89
700	17.61	14.36	9.97	16.84	9.07	15.33
750	19.09	15.56	10.84	18.07	9.85	16.65
800	20.44	16.69	11.59	19.33	10.53	17.84
850	21.75	17.8	12.37	20.58	11.22	19.03
900	23.09	18.84	13.06	21.85	11.88	20.09
950	24.43	19.95	13.87	23.12	12.57	21.36
1000	25.58	20.94	14.48	24.39	13.13	22.43
1100	28.31	23.15	16.04	26.91	14.51	24.85
1200	30.98	25.27	17.5	29.5	15.85	27.26
1300	33.69	27.49	18.99	32.08	17.21	29.63
1400	36.27	29.61	20.4	34.65	18.52	31.95
1500	39.02	31.88	21.99	37.23	19.89	34.45
1600	41.97	34.25	23.56	39.82	21.46	37.08
1700	44.51	36.31	24.95	42.44	22.74	39.22
1800	47.58	38.75	26.76	45.08	24.35	42.33
1900	49.63	40.48	27.76	47.73	25.3	43.89
2000	52.21	42.58	29.17	50.33	26.58	46.25
2200	57.43	46.75	32.05	55.52	29.26	50.62
2400	62.11	50.57	34.6	60.6	31.49	54.74
2600	67.13	54.69	37.4	65.65	34.01	59.16
2800	72.6	59.15	40.53	70.7	36.84	64.1
3000	77.89	63.52	43.52	75.8	39.59	68.96

Figure 4.27: Results of Average Turnaround Times obtained using Exponential Distribution

Figures 4.26 and 4.27 respectively present the results of Average Waiting Time and Average Turnaround Time obtained for the different times of runs and different processes generated using exponential distribution.

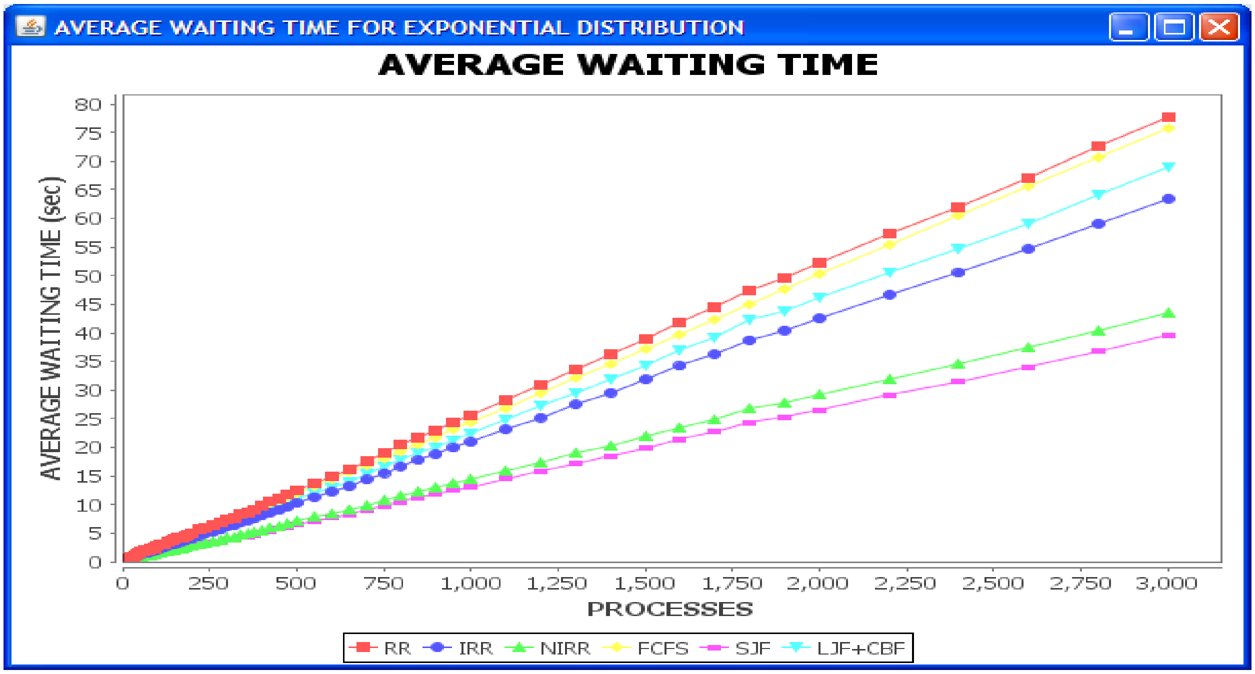


Figure 4.28: Graph of Average Waiting Time using Exponential Distribution

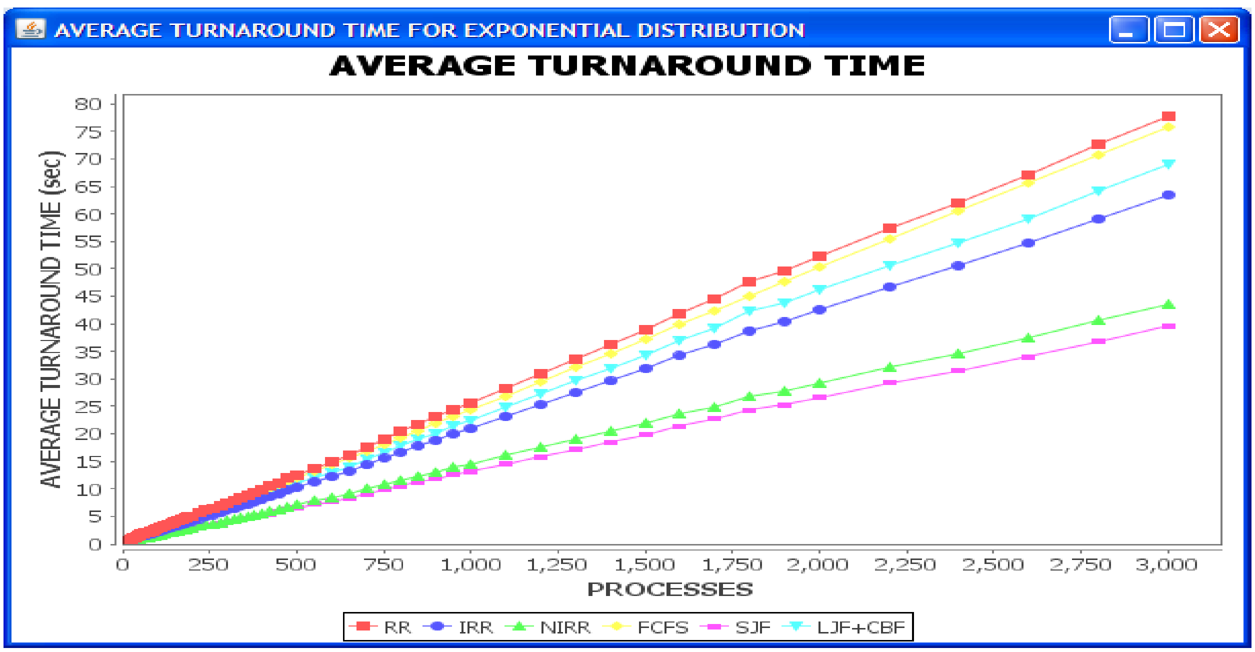


Figure 4.29: Graph of Average Turnaround Time using Exponential Distribution

Figure 4.28 shows the overall graphical result of the Average Waiting Time for all cases of values taken. It was observed from the graph that the SJF produced the best performance, and then the proposed algorithm (NIRR) followed by the IRR, LJF+CBT, FCFS and RR respectively.

Figure 4.29 shows the overall graphical result of the Average Turnaround Time for all cases of values taken. It was observed from the graph that the SJF produced the best performance, and then the proposed algorithm (NIRR) followed by the IRR, LJF+CBT, FCFS and RR respectively.

Average Response Time - Notepad

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	0.03	0.04	0.04	0.04	0.04	0.09
10	0.13	0.09	0.1	0.2	0.1	0.22
15	0.14	0.15	0.19	0.33	0.2	0.35
20	0.24	0.22	0.3	0.48	0.32	0.57
25	0.26	0.28	0.39	0.67	0.4	0.64
30	0.36	0.34	0.47	0.83	0.5	0.77
35	0.38	0.41	0.53	0.99	0.55	0.89
40	0.48	0.48	0.65	1.15	0.69	1.11
45	0.49	0.54	0.72	1.31	0.75	1.23
50	0.59	0.6	0.76	1.46	0.79	1.28
55	0.6	0.67	0.88	1.6	0.9	1.41
60	0.7	0.73	0.92	1.74	0.95	1.53
65	0.71	0.79	0.97	1.87	1.0	1.62
70	0.81	0.85	0.93	1.99	0.98	1.65
75	0.82	0.91	0.98	2.11	1.04	1.71
80	0.91	0.96	1.03	2.22	1.09	1.83
85	0.92	1.02	1.08	2.33	1.15	1.89
90	1.01	1.08	1.13	2.44	1.2	1.95
95	1.02	1.13	1.19	2.54	1.27	2.07
100	1.11	1.19	1.27	2.65	1.35	2.16
110	1.18	1.31	1.39	2.87	1.48	2.33
120	1.32	1.43	1.51	3.09	1.6	2.49
130	1.39	1.55	1.68	3.31	1.79	2.82
140	1.53	1.67	1.75	3.54	1.86	2.96
150	1.6	1.79	1.9	3.77	2.02	3.12
160	1.75	1.91	2.02	3.99	2.15	3.28
170	1.81	2.04	2.09	4.22	2.23	3.5
180	1.96	2.16	2.19	4.43	2.34	3.64
190	2.02	2.28	2.31	4.66	2.48	3.95
200	2.16	2.4	2.41	4.89	2.58	4.11
220	2.33	2.65	2.73	5.35	2.92	4.75
240	2.58	2.89	2.89	5.83	3.12	5.11
260	2.74	3.12	3.05	6.31	3.3	5.45
280	2.98	3.36	3.22	6.76	3.49	5.78
300	3.14	3.59	3.46	7.21	3.78	6.36

Average Response Time - Notepad

300	3.14	3.59	3.46	7.21	3.78	6.36
320	3.37	3.82	3.64	7.67	3.97	6.66
340	3.53	4.05	3.92	8.13	4.27	7.24
360	3.77	4.28	4.09	8.6	4.46	7.62
380	3.93	4.5	4.34	9.06	4.71	7.98
400	4.17	4.73	4.64	9.53	5.05	8.51
425	4.38	5.03	4.95	10.12	5.39	9.15
450	4.67	5.32	5.25	10.72	5.72	9.6
475	4.88	5.61	5.66	11.33	6.17	10.41
500	5.18	5.91	5.93	11.95	6.47	10.99
550	5.65	6.5	6.52	13.18	7.12	11.99
600	6.2	7.08	7.05	14.4	7.68	12.91
650	6.66	7.66	7.52	15.58	8.26	13.85
700	7.2	8.25	8.25	16.79	9.03	15.28
750	7.67	8.83	8.94	18.02	9.8	16.6
800	8.22	9.41	9.55	19.28	10.48	17.79
850	8.68	9.98	10.15	20.53	11.17	18.98
900	9.23	10.55	10.78	21.8	11.83	20.04
950	9.7	11.13	11.37	23.07	12.52	21.31
1000	10.25	11.7	11.88	24.34	13.09	22.38
1100	11.22	12.85	13.12	26.86	14.46	24.8
1200	12.28	13.99	14.34	29.45	15.79	27.21
1300	13.25	15.14	15.6	32.03	17.16	29.58
1400	14.3	16.27	16.8	34.6	18.47	31.9
1500	15.27	17.41	17.98	37.18	19.84	34.4
1600	16.32	18.55	19.52	39.77	21.41	37.03
1700	17.3	19.71	20.69	42.39	22.69	39.16
1800	18.36	20.87	22.07	45.03	24.3	42.28
1900	19.34	22.04	22.98	47.68	25.25	43.84
2000	20.4	23.2	24.16	50.28	26.53	46.2
2200	22.4	25.52	26.64	55.47	29.21	50.57
2400	24.47	27.85	28.57	60.55	31.44	54.69
2600	26.45	30.16	30.85	65.6	33.96	59.11
2800	28.5	32.47	33.39	70.65	36.79	64.05
3000	30.48	34.78	35.92	75.75	39.54	68.91

Figure 4.30: Results of Average Response Times obtained using Exponential Distribution

PROCESS	RR	IRR	NIRR	FCFS	SJF	LJF+CBT
5	10.0	6.0	5.0	4.0	4.0	2.0
10	24.0	17.0	11.0	9.0	9.0	6.0
15	38.0	27.0	16.0	14.0	14.0	10.0
20	60.0	44.0	24.0	19.0	19.0	13.0
25	72.0	53.0	29.0	24.0	24.0	16.0
30	89.0	67.0	35.0	29.0	29.0	19.0
35	99.0	73.0	39.0	34.0	34.0	23.0
40	120.0	90.0	47.0	39.0	39.0	27.0
45	131.0	98.0	52.0	44.0	44.0	31.0
50	139.0	103.0	57.0	49.0	49.0	34.0
55	154.0	114.0	62.0	54.0	54.0	38.0
60	164.0	122.0	68.0	59.0	59.0	42.0
65	174.0	129.0	73.0	64.0	64.0	46.0
70	179.0	134.0	86.0	69.0	69.0	49.0
75	189.0	141.0	92.0	74.0	74.0	52.0
80	199.0	149.0	99.0	79.0	79.0	56.0
85	209.0	156.0	106.0	84.0	84.0	59.0
90	218.0	163.0	112.0	89.0	89.0	62.0
95	230.0	171.0	118.0	94.0	94.0	66.0
100	243.0	179.0	124.0	99.0	99.0	69.0
110	264.0	194.0	138.0	109.0	109.0	76.0
120	284.0	210.0	149.0	119.0	119.0	83.0
130	316.0	234.0	162.0	129.0	129.0	90.0
140	335.0	248.0	175.0	139.0	139.0	96.0
150	358.0	264.0	187.0	149.0	149.0	102.0
160	378.0	277.0	198.0	159.0	159.0	109.0
170	402.0	296.0	209.0	169.0	169.0	115.0
180	423.0	311.0	223.0	179.0	179.0	121.0
190	453.0	336.0	234.0	189.0	189.0	128.0
200	471.0	349.0	246.0	199.0	199.0	135.0
220	533.0	397.0	274.0	219.0	219.0	150.0
240	576.0	431.0	301.0	239.0	239.0	163.0
260	617.0	465.0	326.0	259.0	259.0	176.0
280	655.0	492.0	354.0	279.0	279.0	190.0
300	717.0	542.0	378.0	299.0	299.0	203.0

300	717.0	542.0	378.0	299.0	299.0	203.0
320	751.0	567.0	406.0	319.0	319.0	217.0
340	808.0	611.0	426.0	339.0	339.0	231.0
360	854.0	647.0	451.0	359.0	359.0	244.0
380	897.0	677.0	475.0	379.0	379.0	258.0
400	950.0	717.0	501.0	399.0	399.0	273.0
425	1017.0	767.0	534.0	424.0	424.0	291.0
450	1076.0	809.0	564.0	449.0	449.0	307.0
475	1156.0	871.0	598.0	474.0	474.0	325.0
500	1218.0	918.0	631.0	499.0	499.0	343.0
550	1338.0	1008.0	694.0	549.0	549.0	377.0
600	1445.0	1087.0	756.0	599.0	599.0	412.0
650	1559.0	1172.0	832.0	649.0	649.0	445.0
700	1705.0	1283.0	887.0	699.0	699.0	481.0
750	1842.0	1389.0	955.0	749.0	749.0	519.0
800	1987.0	1504.0	1015.0	799.0	799.0	548.0
850	2114.0	1604.0	1081.0	849.0	849.0	583.0
900	2239.0	1693.0	1141.0	899.0	899.0	617.0
950	2372.0	1797.0	1210.0	949.0	949.0	652.0
1000	2489.0	1890.0	1271.0	999.0	999.0	686.0
1100	2761.0	2098.0	1400.0	1099.0	1099.0	753.0
1200	3022.0	2295.0	1525.0	1199.0	1199.0	821.0
1300	3278.0	2490.0	1651.0	1299.0	1299.0	893.0
1400	3528.0	2681.0	1772.0	1399.0	1399.0	963.0
1500	3798.0	2892.0	1908.0	1499.0	1499.0	1033.0
1600	4085.0	3113.0	2017.0	1599.0	1599.0	1099.0
1700	4331.0	3299.0	2141.0	1699.0	1699.0	1165.0
1800	4649.0	3546.0	2279.0	1799.0	1799.0	1234.0
1900	4853.0	3700.0	2395.0	1899.0	1899.0	1297.0
2000	5114.0	3900.0	2521.0	1999.0	1999.0	1365.0
2200	5616.0	4276.0	2768.0	2199.0	2199.0	1498.0
2400	6084.0	4626.0	3034.0	2399.0	2399.0	1636.0
2600	6583.0	5007.0	3290.0	2599.0	2599.0	1770.0
2800	7113.0	5414.0	3544.0	2799.0	2799.0	1912.0
3000	7626.0	5809.0	3797.0	2999.0	2999.0	2054.0

Figure 4.31: Results of Number of Context Switches obtained using Exponential Distribution

Figures 4.30 and 4.31 respectively present the results of Average Response Time and Number of Context Switches obtained for the different times of runs and different processes generated using exponential distribution.

Figure 4.32 shows the overall graphical result of the Average Response Time for all cases of values taken. It was observed from the graph that the RR has the best performance, IRR came second, then proposed algorithm (NIRR) followed by the SJF, LJF+CBT and FCFS respectively.

Figure 4.33 shows the overall graphical result of the Number of Context Switches for the same processes. It was observed from the graph that LJF+CBT produced the best performance, FCFS and SJF came second producing the same performance followed by the proposed algorithm (NIRR), IRR and RR respectively.

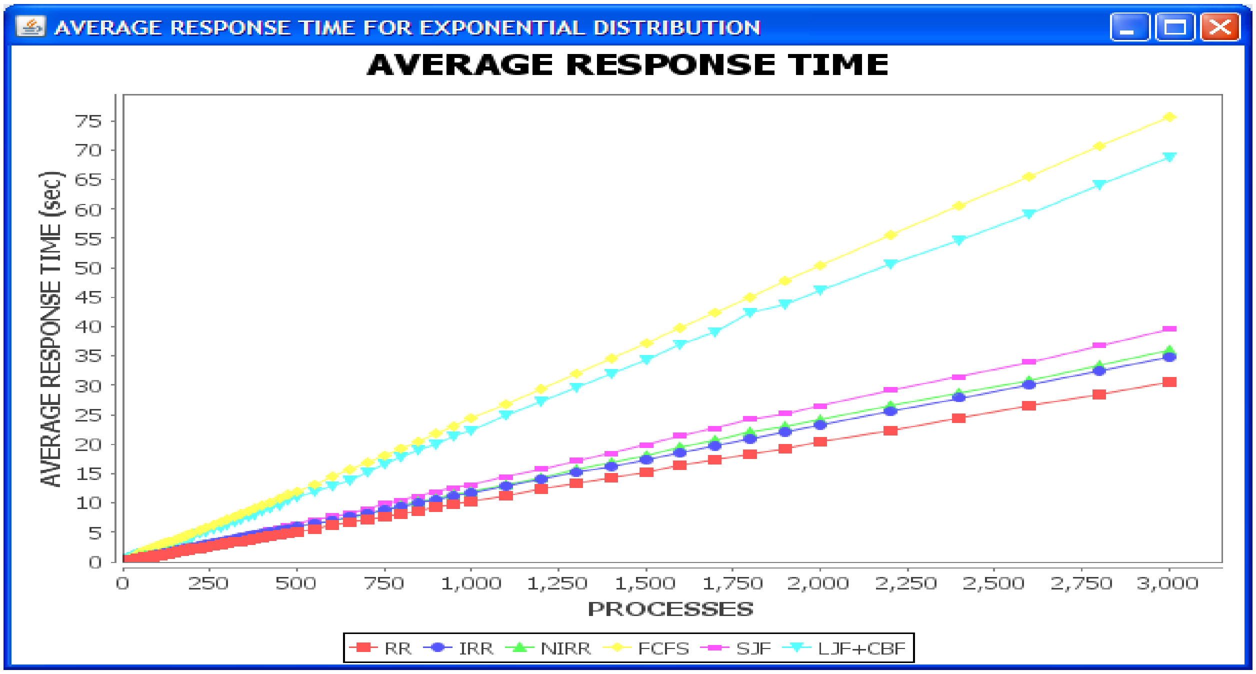


Figure 4.32: Graph of Average Response Time using Exponential Distribution

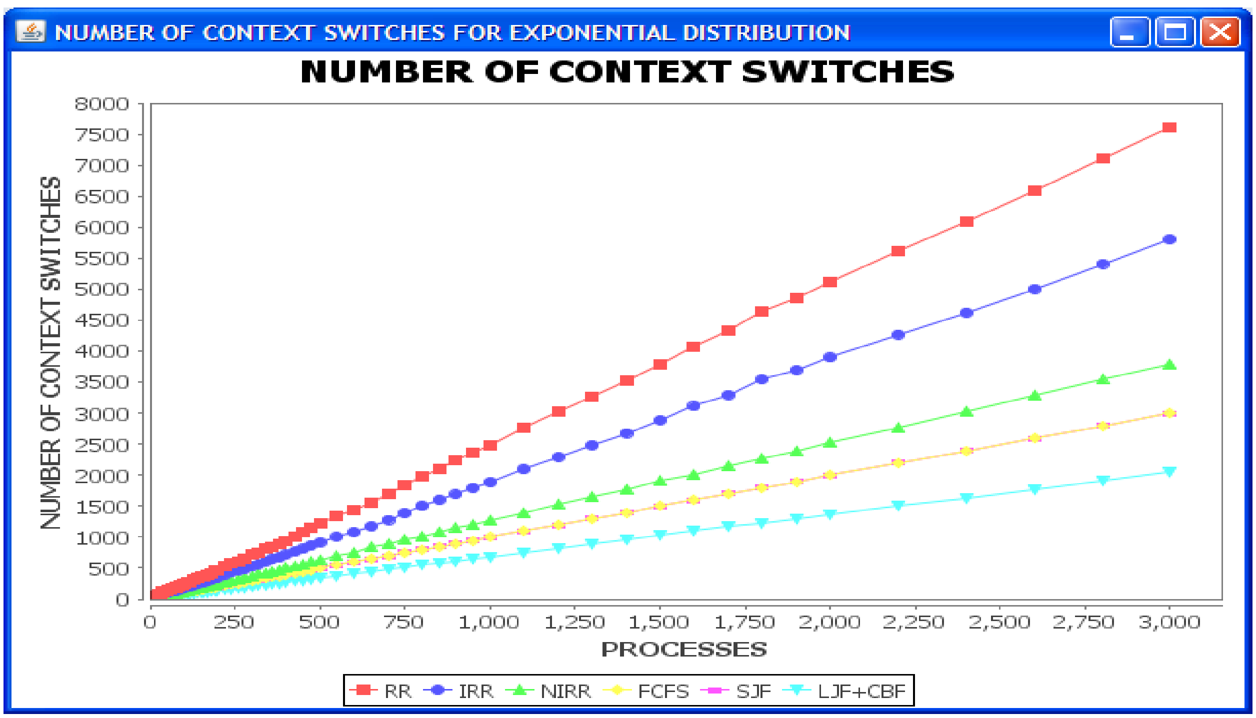


Figure 4.33: Graph of Number of Context Switches using Exponential Distribution

From the results of the five scheduling algorithms (FCFS, SJF, RR, IRR and LJF+CBT) compared with the proposed algorithm (NIRR), it has shown that the proposed algorithm (NIRR) has better Average Waiting Time (AWT) and Average Turnaround Time (ATAT) against IRR, LJF+CBT, FCFS and RR. It also has better Average Response Time (ART) against SJF, LJF+CBT and FCFS, and better Number of Context Switches (NCS) against RR and IRR.

In general, considering the normal distribution, SJF produced the best Average Waiting Time (AWT) and Average Turnaround Time (ATAT) when the number of processes is less or equal to 1500. The proposed algorithm (NIRR) came second, and then followed by LJF+CBT, FCFS, IRR and RR respectively, but as the number of processes exceeds 1500, LJF+CBT tends to produce best Average Waiting Time (AWT) and Average Turnaround Time (ATAT) followed by SJF. The proposed algorithm (NIRR) came third, and then FCFS, IRR and RR respectively. RR produced the best Average Response Time (ART) and IRR came second. When the number of processes is less than 1700, the proposed algorithm (NIRR) came third followed by LJF+CBT, SJF and FCFS respectively. But as number of processes exceeds 1700, LJF+CBT came third followed by the proposed algorithm (NIRR), SJF and FCFS respectively. While LJF+CBT produced the best Number of Context Switches (NCS), and then FCFS and SJF came second producing the same performance followed by the proposed algorithm (NIRR), IRR and RR respectively.

With the uniform distribution, SJF produced the best Average Waiting Time (AWT) and Average Turnaround Time (ATAT), the proposed algorithm came second followed by LJF+CBT, FCFS, IRR and RR respectively. RR produced the best Average Response Time (ART), IRR came second, and then the proposed algorithm (NIRR) followed by SJF, LJF+CBT and FCFS respectively. While LJF+CBT produced the best Number of Context Switches (NCS), and then FCFS and SJF came second producing the same performance followed by the proposed algorithm (NIRR), IRR and RR respectively (Abdulrazaq *et al*, 2014).

And with the exponential distribution, SJF produced the best Average Waiting Time (AWT) and Average Turnaround Time (ATAT), the proposed algorithm came second followed by IRR, LJF+CBT, FCFS, and RR respectively. RR produced the best Average Response Time (ART), IRR came second, and then the proposed algorithm (NIRR) followed by SJF, LJF+CBT and FCFS respectively. While LJF+CBT produced the best Number of Context Switches (NCS), and then FCFS and SJF came second producing the same performance followed by the proposed algorithm (NIRR), IRR and RR respectively.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

This chapter presents the summary, conclusion and recommendation of this thesis work.

5.1 Summary

CPU is one of the primary computer resources, so its scheduling is essential to an operating system's design. CPU scheduling is important because it plays an important role in effective resource utilization and the overall performance of the system. There exists a number of CPU scheduling algorithms like First-Come-First-Serve, Shortest-Job-Scheduling, Priority CPU scheduling and Round Robin scheduling, but due to a number of disadvantages these algorithms have, they are rarely used except Round Robin scheduling in timesharing operating systems. The performance of Round Robin Scheduling is sensitive to time quantum selection, because if time quantum is very large then it is the same as the First-Come-First-Serve Scheduling, and if the time quantum is extremely too small then it is the same as Processor sharing algorithm. Each value 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.

5.2 Conclusion

This thesis presents an algorithm that is based on the modification of the Improved Round Robin scheduling algorithm by Manish and AbdulKadir (2012). Its goal is to increase the performance of the Round Robin CPU scheduling algorithm by reducing the average waiting time, number of context switch, average response time and average turnaround time. This proposed algorithm (NIRR) together with FCFS, SJF, RR, IRR and LJF+CBT CPU scheduling algorithms were implemented in Java and their results were compared based on AWT, ATAT, ART and NCS for different categories of processes that were generated randomly (i.e. using normal, uniform or exponential distribution for burst time and Poisson for arrival time).

The simulation results show that with normal distribution, when the number of processes is less than or equal to 1500, SJF will be the best scheduling algorithm in terms of minimizing AWT and ATAT followed by the proposed algorithm (NIRR), LJF+CBT, FCFS, IRR and RR respectively. But as number of processes exceeds 1500, LJF+CBT will be the best scheduling algorithm, and then it will be followed by SJF, the proposed algorithm (NIRR), FCFS, IRR and RR respectively. LJF+CBT will be the best scheduling algorithm in terms of minimizing NCS, and then FCFS and SJF will come second because they will produce the same performance, this will be followed by the proposed algorithm (NIRR), IRR and RR respectively. And RR will be the best algorithm in terms of minimizing ART, and then followed by IRR, when the number of processes is less than or equal to 1700, the proposed algorithm (NIRR) will come third followed by LJF+CBT, SJF and FCFS respectively. But as number of processes exceeds 1700, LJF+CBT will come third followed by the proposed algorithm (NIRR), SJF and FCFS respectively.

With uniform distribution, SJF will be the best scheduling algorithm in terms of minimizing AWT and ATAT, and then it will be followed by the proposed algorithm (NIRR), LJF+CBT, FCFS, IRR and RR respectively. LJF+CBT will be the best scheduling algorithm in terms of minimizing NCS, and then FCFS and SJF will come second because they will produce the same performance, this will be followed by the proposed algorithm (NIRR), IRR and RR respectively. And RR will be the best algorithm in terms of minimizing ART, and then followed by IRR, the proposed algorithm (NIRR), SJF, LJF+CBT and FCFS respectively.

And with the exponential distribution, SJF will be the best scheduling algorithm in terms of minimizing AWT and ATAT, and then it will be followed by the proposed algorithm (NIRR), IRR, LJF+CBT, FCFS and RR respectively. LJF+CBT will be the best scheduling algorithm in terms of minimizing NCS, and then FCFS and SJF will come second because they will produce the same performance, this will be followed by the proposed algorithm (NIRR), IRR and RR respectively. And RR will be the best algorithm in terms of minimizing ART, and then followed by IRR, the proposed algorithm (NIRR), SJF, LJF+CBT and FCFS respectively.

Based on the results obtained, it was observed that the performance of the proposed algorithm (NIRR) is better than that of the Simple Round Robin CPU scheduling algorithm and the Improved Round Robin CPU scheduling algorithm by Manish and AbdulKadir (2012), in the sense that, it produces minimal average waiting time, average turnaround time and number of context switches.

5.3 Recommendation

The recommendation in this work is to implement the proposed algorithm (i.e. A New Improved Round Robin (NIRR) CPU scheduling algorithm) in the systems that adopt the Round Robin scheduling; so as to improve the performance of the systems. Aside from the improvements in the performance of Round Robin CPU scheduling algorithm, as for the main line of future research as related to this work, it is suggested that the following areas need to be looked into:

1. A test for more performance criteria (i.e. CPU utilization and throughput time), this can be achieved by taking into consideration context switch overhead time.
2. Also, since there are so many works on improving Round Robin CPU scheduling algorithm, assessment of these algorithms with the proposed algorithm by considering

both I/O and CPU bound processes should be done, so as to have a best Round Robin scheduling algorithm that will be used in the systems that adopt Round Robin scheduling, in order to achieve the best performance of the systems.

REFERENCES

Abbas, N., Ali, K., and Seifedine, K. (2011). A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average. *International Journal of Computer Science Issues , 8* (4), 224-229.

Abdullahi, I and Junaidu, S. B. (2013). Empirical Framework to Migrate Problems in Longer Job First Scheduling Algorithm (LJF+CBT). *International Journal of Computer Applications , 75* (14), 9-14.

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

Abdulrazaq, A., Salisu, A., Ahmad, M. M., and Saleh, E. A. (2014). An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering , 4* (2), 601-610.

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), 2382-2385.

Behera, H.S, Brajendra, K. S, Anmol, K. P and Gangadhar, 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.

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 (0975 – 8887) , 5* (5), 10-15.

Chapter 6: Queuing systems. (n.d.). Retrieved September 15, 2014, from www.emse.fr/~xie/SJTU/Ch6Queue.ppt.

CPU Scheduling. (n.d.). Retrieved October 8th, 2013, from http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html

CPU Scheduling. (n.d.). Retrieved October 31, 2013, from http://www.scs.stanford.edu/07au-cs140/notes/l5.pdf.

Dibyendu, B and Iti, S. (2013). Dynamic Time Quantum in Shortest Job First Scheduling Algorithm (DTQSJF) for Unpredictable Burst Time of Processes. *International Journal of Computer Science & Engineering Technology , 4* (3), 208-212.

Elements of Queuing Systems. (n.d.). Retrieved September 16, 2014, from www.staff.um.edu.mt/jskl1/simweb/intro.htm.

Frédéric, H. (2007). *www.it.uu.se/edu/course/homepage/oskomp/vt07/lectures/scheduling_algorithms/handout.pdf*. Retrieved February 28, 2014

Insup, L. (2007). Retrieved September 16, 2014, from www.cis.upenn.edu/~lee/07cis505/Lec/os-schedule-v2.pdf.

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.

Ivo, A., and Jacques, R. (2002, February 28). Queueing Theory. Eindhoven, The Netherlands.

Jerry, B, John, S.C, Barry, L.N and David K. (2005). *Discrete-Event System Simulation* (Fourth ed.). Pearson Education International.

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), 338-354.

Lalit, K, Rajendra, S and Praveen, S. (2011). Optimized Scheduling Algorithm. *International Journal of Computer Applications* , 106-109.

Little, D. C. (1961). A Proof for the Queuing Formula: $L= \lambda W$. *Operations Research , 9* (3), 383-387.

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

Mohammed, A. H. (2011). A selective Quantum of Time for Round Robin Algorithm to Increase CPU Utilization. *International Journal of Computer Information Systems , 3* (2), 54-59.

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

Prashant, M., and Gopalkrushna, P. (2012). Minimizing the Waiting Time for Service with Queuing Model. *National Monthly Refereed Journal of Research in Science and Technology , 1* (11), 12-19.

Queuing System Characteristics. (n.d.). Retrieved September 22, 2014, from www.nptel.ac.in/courses/110106046/Module%209/Lecture%202.pdf.

Rahul, S., and Umesh, C. J. (2012). Process Scheduling Using Repeatative Subtraction (An algorithm dependent on the number of processes executed since its arrival). *International Journal of Advanced Research in Computer Science and Software Engineering , 2* (11), 425-429.

Rakesh, M, Manas, D, Lakshmi, M. P and Sudhashree. (2011). Design and Performance Evaluation of A New Proposed Fittest Job First Dynamic Round Robin (FJFDRR) Scheduling Algorithm. *International Journal of Computer Information Systems , 2* (2), 23-27.

Ramakrishna, M., and Pattabhi, G. R. (2013). Efficient Round Robin CPU Scheduling Algorithm for Operating Systems. *International Journal of Innovative Technology and Research , 1* (1), 103-109.

Rashid, M. (2014). *First-Come-First-Served (FCFS) Scheduling*. Retrieved September 18, 2014, from http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/fcfsSchedule.htm

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

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

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

Soraj, H and Roy, K.C. (n.d.). *Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice*. Retrieved December 10th, 2012, from International Journal of Data Engineering: www.cscjournals.org/csc/manuscript/Journals/IJDE/.../IJDE-57.pdf

Supriya. R, Reena. D and Smita. R. (2012). An Optimum Time Quantum using Linguistic Synthesis for Round Robin CPU Scheduling Algorithm. *International Journal on Soft Computing , 3* (1), 57-66.

Surendra, P. V. (2013). A Finest Time Quantum for Improving Shortest Remaining Burst Round Robin (SRBRR) Algorithm. *Journal of Global Research in Computer Science , 4* (3), 10-15.

Suri, P.K and Sumit, M. (2012). Design of Stochastic Simulator for Analyzing the Impact of Scalability on CPU Scheduling Algorithms. *International Journal of Computer Applications (0975 – 8887) , 49* (17), 4-9.

Types Of Scheduling. (2010). Retrieved February 28, 2014, from www.go4expert.com/articles/types-of-scheduling-t22307/.

Virtamo, J. (n.d.). *Queuing Systems.* Retrieved September 22, 2014, from www.netlab.tkk.fi/opetus/s383143/kalvot/E_priority.pdf.

William, S. (2012). *Operating Systems: Internals and Design Principles* (Seventh ed.). New Jersey: Pearson Education, Inc.

APPENDIX

```
public double [][] NewImprovedRoundRobin(int []burst_value,int []arrival_value,int num){
    int Cont_Switch;
    double Av_WT,Av_TAT,Av_RES;
    Boolean state = false;
    double [][] more=new double[7][num+1];
Thesis_Manipulation op[] = new Thesis_Manipulation[num+1];       // creatng array of objects
for(int i=0;i<=num;i++)
    op[i]= new Thesis_Manipulation();
int index [] = new int[num+1];
    int sum=0;
    int t=0;
for(int m=1;m<=l;m++)
{
op[m].pr=m;
op[m].bt=burst_value[m];
op[m].at=arrival_value[m];
sum+=op[m].bt;
op[m].copy_of_bt=op[m].bt;
}
double tq,ref_point=0,total=0;
int indexx=1, response = 0, context=0,initial=1,process_number=0,
        count,end=0;
while(t<sum)
{
//    start of tq calculation
  //    start of tq calculation
if (indexx==1){
    tq=op[indexx].bt;
    process_number = process_number+1;
    process_number=0;
    index[indexx] = op[indexx].pr;
    op[indexx].rt = response-op[indexx].at;
    t+=op[indexx].bt;
    ref_point=t;
    response=t;
    op[indexx].bt-=op[indexx].bt;
    op[indexx].tat=t-op[indexx].at;
    op[indexx].wt=op[indexx].tat-op[indexx].copy_of_bt;
    indexx = indexx +1;
```

```
    context=context+1;
}
else{
    indexx = 2;
        for(int k=2;k<=l;k++)
         {
         if((op[k].bt>0)&&(op[k].at<=ref_point))
                         {
            total=total+op[k].bt;
            process_number = process_number+1;
             }
}
        tq = Math.ceil(total/process_number);
        process_number=0;
        total = 0;
    int temp;
            for(int i=2;i<l;i++)
                      {
            for(int k=2;k<l;k++)
                        {
                if((op[k+1].bt < op[k].bt)&&(op[k+1].bt>0)&&(op[k].bt>0))
                        {
          temp = op[k+1].bt;
          op[k+1].bt = op[k].bt;
          op[k].bt = temp;
          temp = op[k+1].at;
          op[k+1].at = op[k].at;
          op[k].at = temp;
          temp = op[k+1].copy_of_bt;
          op[k+1].copy_of_bt = op[k].copy_of_bt;
          op[k].copy_of_bt = temp;
          temp = op[k+1].pr;
          op[k+1].pr = op[k].pr;
          op[k].pr = temp;
                  }
                  }
                   }
    for(int j=2;j<=l;j++)
                      {
        if((op[j].bt>0)&&(op[j].at<=ref_point)){
```

```
    //            response time calculations start here
for (int i =2; i < index.length; i++){
                  if (index[i]==op[j].pr){
                      state =true;
                    break;
                  }
                  else{
                      state = false;
                  }
              }
              if (state==false){
                      index[j] = op[j].pr;
                      op[j].rt = response-op[j].at;
                 }
//           response time calculations ends here
    if((op[j].bt>tq)&&(op[j].bt<=(1.5*tq)))
              {
              t+=op[j].bt;
              response=t;
              op[j].bt-=op[j].bt;
              ref_point=t;
              op[j].tat=t-op[j].at;
              op[j].wt=op[j].tat-op[j].copy_of_bt;
              context=context+1;
                }
              else if((op[j].bt>tq)&&(op[j].bt>(1.5*tq)))
              {
                  op[j].bt-=tq;
                  t+=tq;
                  response=t;
                  ref_point=t;
                  context=context+1;
              }
          else
              {
              t+=op[j].bt;
              response=t;
              ref_point=t;
              op[j].bt=0;
              op[j].tat=t-op[j].at;
```

```
                op[j].wt=op[j].tat-op[j].copy_of_bt;
                context=context+1;
                }
    }
}
}
}
int tot_turn=0,tot_wait=0,tot_res=0;
for(int i=1;i<=l;i++)
    {
    tot_turn+=op[i].tat;
    tot_wait+=op[i].wt;
    tot_res+=op[i].rt;
    }
double avg_turn=(float)tot_turn/(1000*l);
double avg_wait=(float)tot_wait/(1000*l);
double avg_res=(float)tot_res/(1000*l);
String Anomm=String.valueOf(String.format("%.2f ",+avg_turn));
String Anom=String.valueOf(String.format("%.2f ",+avg_wait));
String Ano=String.valueOf(String.format("%.2f ",+avg_res));
Cont_Switch=context-1;
Av_TAT = Double.parseDouble(Anomm);
Av_WT = Double.parseDouble(Anom);
Av_RES = Double.parseDouble(Ano);
more[1][l]=l;
more[2][l]=Av_WT;
more[3][l]=Av_TAT;
more[4][l]=Av_RES;
more[5][l]=Cont_Switch;
}
    return more;
 }
```