IMPROVED ROUND ROBIN WITH HIGHEST RESPONSE RATIO NEXT (IRRHRRN) CPU SCHEDULING ALGORITHM

BY

MUHAMMAD ABUBAKAR BABA'ABU

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY,

ZARIA, NIGERIA

NOVEMBER, 2016.

IMPROVED ROUND ROBIN WITH HIGHEST RESPONSE RATIO NEXT (IRRHRRN) CPU SCHEDULING ALGORITHM

BY

MUHAMMAD ABUBAKAR BABA'ABU, B.TECH MATHEMATICS WITH COMPUTER SCIENCE, (F.U.T MINNA), 2004

P15SCMT8002

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

NOVEMBER, 2016

DECLARATION

I declare that the work in this project report entitled Improved Round Robin with Highest Response Ratio Next (IRRHRRN) CPU Scheduling Algorithm has been performed by me in the Department of Mathematics under the supervision of Dr. S.E Abdullahiand Prof. O. S. Adewale. 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.

MUHAMMAD ABUBAKAR BABA'ABU		
Name of student	Signature	Date

CERTIFICATION

This thesis entitled IMPROVED ROUND ROBIN WITH HIGHEST RESPONSE RATIO NEXT (IRRHRRN) CPU SCHEDULING ALGORITHM by (AbubakarBaba'abu, MUHAMMAD)meets the regulation governing the award of the degree of M.Sc Computer Science of Ahmadu Bello University, Zaria 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

This thesis is dedicated to my beloved parents Late Alhaji Muhammad S. Lemu and Hajiya Fatima Muhammad Alkali for supporting me through life both morally and financially. It is also dedicated to the entire family of Muhammad S. Lemu.

ACKNOWLEDGEMENTS

First, I acknowledge my supervisor (major) Dr. S. E. Abdullahi for his advice, corrections and time spent on seeing to the completion of this thesis and my supervisor (minor) Prof. O. S. Adewale for his contributions.I also acknowledge the effortsof the entire lecturers of the Department of Mathematics for their academic and intellectual support towards the success of this thesis.

A special acknowledgment also goes to my parents lateAlh. Muhammad S. Lemu and Hajiya Fatima Muhammad Alkali, my brothers Mall.Aliyu Muhammad Lemu and late Dr. Muhammad Hassan, and my entire family membersfor their financial, moral and courageous support throughout my entire study.

I cannot forget mentioning some of my colleagues and friends who contributed in one way or another towards thesuccessful completion of this thesis. These people include Mal.AliyuSalis, Mal. AbdulrazakAbdulrahim, Mal. Suleiman Ahmad Sabo, Mal. Hussaini and my entire classmates.

Finally, a special acknowledgment also goes to the Niger State Ministry of Finance for granting me the approval to run this program.

ABSTRACT

Round Robin CPU scheduling algorithm is the most suitable scheduling algorithm used in timesharing operating systems, but its performance is sensitive to time quantum selection, which is the same as the First-Come-First-Serve (FCFS) Scheduling or Processor sharing algorithm if the time quantum is large or extremely too small. This thesis proposed an algorithm known as Improved Round Robin with Highest Response Ratio Next (IRRHRRN) CPU Scheduling Algorithmwhich made improvements on the Improved Round Robin scheduling algorithms by adopting the principle of Highest Response Ratio Next (HRRN) in static Round Robin implementation. The proposed algorithm (IRRHRRN) together with Round Robin (RR),FCFS, Improved Round Robin (IRR) and An Additional Improvement in Round Robin (AAIRR) CPU scheduling algorithms were implemented in Java and their results were compared based on Average Waiting Time (AWT), Average Turnaround Time (ATAT), Average Response Time (ART) and Number of Context Switches (NCS) for different categories of processes that were generated randomly (i.e. using uniform distribution for burst time and Poisson for arrival time). It was observed that this algorithmis the best scheduling algorithm in terms of minimizing AWT, ATAT and ART based on the results obtained.It is therefore recommendedthat this algorithm is implemented in systems which adopt the Round Robin scheduling whose time quantum are provided statically (user inputs the time quantum before the execution of the processes); so as to improve the performance of the systems.

TABLE OF CONTENTS

DECLARATION ii
CERTIFICATION iii
DEDICATION iv
ACKNOWLEDGEMENTS v
ABSTRACT vi
TABLE OF CONTENT vii
LIST OF TABLES x
LIST OF FIGURES xi
LIST OF ABBREVIATIONS xii
CHAPTER ONE 1
INTRODUCTION 1
1.1 Background to the study 1
1.2 Research Motivation 3
1.3 Research aim and objectives 4
1.4 Research Methodology 5
1.5 Contribution to Knowledge 5
CHAPTER TWO 7
LITERATURE REVIEW 7
2.1 Multiprogramming 7
2.2 Processes 8
2.2.1 CPU-I/O burst cycle 10
2.3 Scheduling 11
2.3.1 Schedulers 11
2.3.2 Long-term Scheduler/admission Scheduler 12
2.3.3 Mid/Medium-term Scheduler 13
2.3.4 Short-term Scheduler/CPU scheduler 14
2.3.5 Dispatcher 14
2.4 CPU scheduling algorithms 15
2.4.1 First-Come First-Serve (FCFS) 15
2.4.2 Shortest-Job-First (SJF) 15

2.4.3 Priority Scheduling (PS) 16
2.4.4 Round Robin Scheduling (RR) 17
2.5 Scheduling criteria 19
2.6 Related Work 20
CHAPTER THREE 27
DESIGN OF THE PROPOSED ROUND ROBIN CPU SCHEDULING ALGORITHM 27
3.1 The Proposed Improved Round Robin with Highest Response Ratio Next (IRRHRRN) Scheduling Algorithm 27
3.1.1 The pseudo code of the proposed Improved Round Robin with Highest Response Ratio Next (IRRHRRN) Scheduling Algorithm 28
3.2 How the Proposed Algorithm works 322
3.3 Illustrative Examples 35
3.3.1 Round Robin (RR) 366
3.3.2 First Come First Serve (FCFS) 377
3.3.3 Improved Round Robin (IRR) 388
3.3.4 An Additional Improvement in Round Robin (AAIRR) 400
3.3.5 Round Robin with Highest Response Ratio Next (RRHRRN) Scheduling Algorithm 411
3.3.6 Improved Round Robin With Highest Response Ratio Next (IRRHRRN) CPUScheduling Algorithm 422
3.4 System Architecture 444
CHAPTER FOUR 466
IMPLEMENTATION OF THE PROPOSED IMPROVED ROUND ROBIN WITH HIGHEST RESPONSE RATIO CPU SCHEDULING ALGORITHM 466
4.1 Assumption 466
4.2 System Requirements 477
4.2.1 Experimental setup 477
4.1 Results Discussion 488
CHAPTER FIVE 579
SUMMARY, CONCLUSION AND RECOMMENDATION 599
5.1 Summary 599
5.2 Conclusion 60

5.3 Recommendation..... 60

LIST OF TABLES

Table 3.1: Process Table...……..**Error!**

Bookmark not defined..33

Table 3.2: Process Table 2…….…….35

Table 3.3: Comparative Table……..43

Table 4.1: Process Identity…….….50

Table 4.2: Some of the Results of Average Waiting Times……....51

Table 4.3: Some of the Results of Average Turnaround Times…….....52

Table 4.4: Some of the Results of Average Response Times…….....54

Table 4.5: Some of the Results of Number of Context Switches……..……..55

Table 4.6: Performance Percentage Comparing the Algorithms..…….……..57

LIST OF FIGURES

Figure 2.1: Process state diagram…….8

Figure 2.2: Diagram illustrating Alternating Sequence of CPU and I/O Burst……..10

Figure 2.3 Queuing diagram of a process scheduling……12

Figure 2.4: Mid-term scheduler……..13

Figure 3.1: Gantt chart representation of the illustrated example for the Proposed Algorithm…….34

Figure 3.2: Gantt chart representation of RR……..36

Figure 3.3: Gantt chart representation of FCFS……..37

Figure 3.4: Gantt chart representation of IRR……39

Figure 3.5: Gantt chart representation of AAIRR……40

Figure 3.6: Gantt chart representation of RRHRRN……41

Figure 3.7: Gantt chart representation of IRRHRRN……..42

Figure 3.8: The System Architecture……...44

Figure 4.1: System Interface with some process execution in progress……..….49

Figure 4.2: Graph of Average Waiting Time……...51

Figure 4.3: Graph of Average Turnaround Time……53

Figure 4.4: Graph of Average Response Time……..…..54

Figure 4.5: Graph of Number of Context Switches…….....56

LIST OF ABBREVIATIONS

AAIRR: An Additional Improvement in Round Robin

ART: Average Response Time

ATAT: Average Turn-Around Time

AWT: Average Waiting Time

CPU: Central Processing Unit

CS: Context Switch

FCFS: First Come First Served

HRR: Highest Response Ratio

HRRN: Highest Response Ratio Next

I/O: Input Output

IRR: Improved Round Robin

IRRHRRN: Improved Round Robin with Highest Response Ratio Next

NCS: Number of Context Switches

OS: Operating System

RR: Round Robin

RRHRRN: Round Robin with Highest Response Ratio Next

RT: Response Time

SJF: Shortest Job First

TAT: Turn-Around Time

TQ: Time Quantum

WT: Waiting Time

CHAPTER ONE

INTRODUCTION

This chapter discusses the introductory part of this thesis which includes the background to the study, research motivation and goals which states the research questions for which the thesis should provide answers to, the research aim and objectives, the methodology that will be used to answer those questions and finally the summary of the thesis contribution to knowledge.

1.1 Background to the study

An operating system (OS) 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. It is the most important program needed for starting up and using the hardware, which has different managing tasksеach of which is performed by one of the management units (e.g. memory, process, storage management units) of the OS. The process management, as one of these management units, allocates the processor to the processes using several allocating algorithms (Saeidiand Hakimeh, 2012).The basic idea is to keep the CPU busy as much as possible by executing a process until it must wait for an event, and then switch to another process (Suri and Sumit, 2012). A single user cannot, in general keep either the CPU or the input/output (I/O) devices busy at all times. In multiprogramming systems, when there is more than one runable 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.

The basic CPU scheduling algorithms are First ComeFirstServe (FCFS), Shortest Job FirstScheduling (SJF), Round Robin (RR) and Priority scheduling (PS), 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 system; and considered most widely used scheduling algorithms (Ajit*et al.,* 2010;Saeidi and Hakimeh, 2012; Behera*et al.,* 2012;Ishwariand Deepa, 2012; Sorajand Roy, 2011).

The performance of Round Robin Scheduling is sensitive to time quantum selection, because if time quantum is very large then Round Robin scheduling is the 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; Sorajand Roy,2011). 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, through/put, CPU utilization and number of context switch (Abdulrahim*et al.,* 2014).

Time quantum provision in Round Robin CPU Scheduling is done in two different categories (Abdulrahim*et al.*, 2014).

1. Static provision of time quantum: this is the method in which the time quantum to be used by the processes is predefined usually in 10-100*ms* range(Saeidi and Hakimeh, 2012). Here the time quantum is provided without putting into consideration the process burst time or the processes properties to which it will be applied.
2. Dynamic determination of the time quantum: this is the method in which the time quantum to be used on processes is determined by the properties of the processes on which it will be used. Properties like processes burst time, arrival time etc. The aim of dynamic determination of time quantum in the Round Robin CPU Scheduling algorithm is to improve on its performance.

This thesis shall concentrate only on improvements made on static determination of time quantum in Round Robin Scheduling algorithms.

1.2 Research Motivation

So many improvements have been made in the area of increasing the performanceof Round Robin CPU Scheduling algorithm in static implementation of time quantum, which in turn increase the performance of the improved algorithms compared to the traditional Round Robin CPU Scheduling algorithm.

Manish and AbdulKadir (2012) presented an algorithm which drastically minimizes waiting time and turnaround time with respect to the simple Round Robin scheduling algorithm.It does this by reallocating the CPU to the currently running process if its remaining burst time is less than the time quantum. But, the problem it may encounter is, when the time quantum assigned to the processes is greater than or equals to the half of the maximum value of the processes burst time, then the algorithm will be the same as First Come First Serve (FCFS) CPU scheduling algorithm, which is one of the main drawback of Round Robin CPU scheduling algorithm.Abdulrahim*et al.*, (2014) tried to solve the problem that may be encountered by the algorithm as proposed byManish and AbdulKadir (2012) by using the principle of Shortest Job First (SJF) CPU scheduling algorithm. But in this algorithm, processes with longer burst time will have to stay for a long period of time before they are executed. This tends to violate the fair nature of Round Robin CPU scheduling algorithm.

The goal of this thesis is to add improvements to the Improved Round Robin CPU scheduling algorithms by Manish and AbdulKadir(2012) and Abdulrahim*et al.*, (2014) by adopting the principle of Highest Response Ratio Next (HRRN) thereby eliminating the problem of shorter

jobs gaining more priority than longer jobs that could lead to longer jobs being starved. Highest Response Ratio Next (HRRN) scheduling is a non-preemptive discipline similar to Shortest Job Next (SJN) in which the priority of each process is dependent on its estimated service time or burst time and also the amount of time it has spent waiting. This will be used in order to increase the performance of the Round Robin scheduling algorithm by affecting the average waiting time, average turnaround time, average response time and number of context switch positively.

1.3 Research aim and objectives

The aim of this work is to develop an algorithm that will make improvements to the RR CPU scheduling algorithm. This will be done by adding improvements to the Improved Round Robin CPU scheduling algorithm (IRR) by Manish and AbdulKadir(2012) and An Addition Improvement in Round Robin CPU scheduling algorithm (AAIRR) byAbdulrahim*et al.*, (2014).The following are the objectives of this research work:

1. To study the Improved Round Robin scheduling algorithms that statically determines the time quantum.
2. To develop an algorithm that makes improvements on theImproved Round Robin scheduling algorithms by adopting the principle of Highest Response Ratio Next (HRRN)approach.
3. To implement of the developed Round Robin CPU scheduling algorithm.
4. To evaluate of the five implementations (i.e. simple RR, FCFS, IRR, AAIRR and the improved algorithm).

1.4 Research Methodology

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

1. Intensive study of CPU scheduling algorithms
2. Review of literatures on Round Robin scheduling algorithm
3. Develop an algorithm to improve the IRR and AAIRR CPU scheduling algorithms
4. Simulate the improved CPU scheduling algorithm, RR, FCFS, IRR andAAIRR using Java programming
5. The time quantum shall be determined statically (it will be determined by the user/designer before the execution of the processes). The number of processes shall also be determined by the user.
6. The data to be used in the simulation will be generated randomly: the burst time of the processes will be generated using uniform distribution and the arrival time will be generated using exponential distribution
7. The algorithms (i.e. RR, FCFS, IRR, AAIRR and the improved algorithm) will be compared based on Average Waiting Time, Average Turnaround Time, Average Response Time and Number of Context Switch for the different categories of burst times generated

1.5 Contribution to Knowledge

The thesis will contribute to the improvement of the RR scheduling algorithm by:

1. Developing an algorithm that will make improvements to the Improved Round Robin CPU scheduling algorithms by Manish and AbdulKadir (2012) and Abdulrahim*et al*., (2014) by adopting the principle of Highest Response Ratio Next (HRRN), which will in turn increase the performance of the RR scheduling algorithm by affecting the average

waiting time, average turnaround time, average response time and number of context switches positively.

2. It also solves the issue of starvation that may be encountered by the algorithm by Abdulrahim*et al.,* (2014) especially when there are continuous streams of Shorter Jobs coming into the ready queue.

CHAPTER TWO

LITERATURE REVIEW

This chapter discusses the concepts of multiprogramming, processes, scheduling, scheduling algorithms, scheduling criteria and survey of works done in the field of static provision of time quantum in Round Robin CPU scheduling algorithm. It also discusses theRound Robin with Highest Response Ratio Next (RRHRRN) scheduling algorithm

2.1 Multiprogramming

One of the most important aspects of operating systems is the ability to multiprogram. 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 have one to execute.

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.

In a non-multiprogrammed (single processing) system, 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.

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 and awaiting allocation of the 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. The process of making this selection 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 among them. The process of making this selection is called CPU scheduling.

2.2 Processes

Scheduling refers to the way processes are assigned to run on available Central Processing Unit (CPU) (Silberschatz*et al.,* 2006). Each Process (a program in execution) passes through some stages in their execution and allocation to CPU as will be illustrated in process-state diagram. A Process is started at arrival time which gets into the Ready Queue based on some scheduling algorithm and waits for the Job scheduler/ dispatcher which decide on what scheduling algorithm to use. The process state diagram of Fig 2.1 illustrates this.

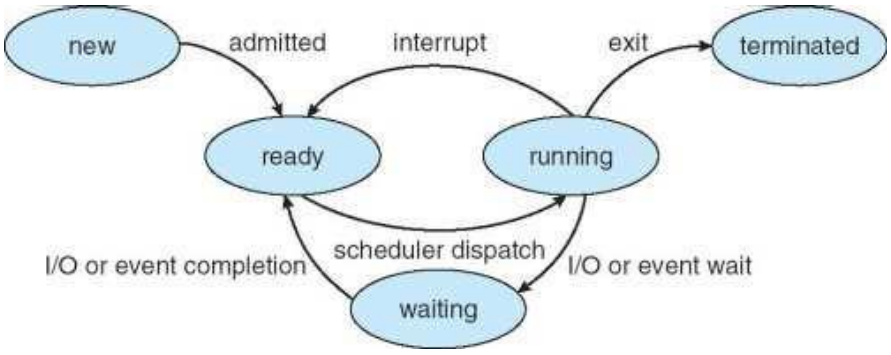


Figure 2.3: Process state diagram

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

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

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 a scheduler switches processes in this way, 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.

2.2.1 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 continue until the execution of the process is completed (as shown in Figure 2.2). 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 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 we do not want to leave the CPU idle while waiting for the I/O operation to finish.

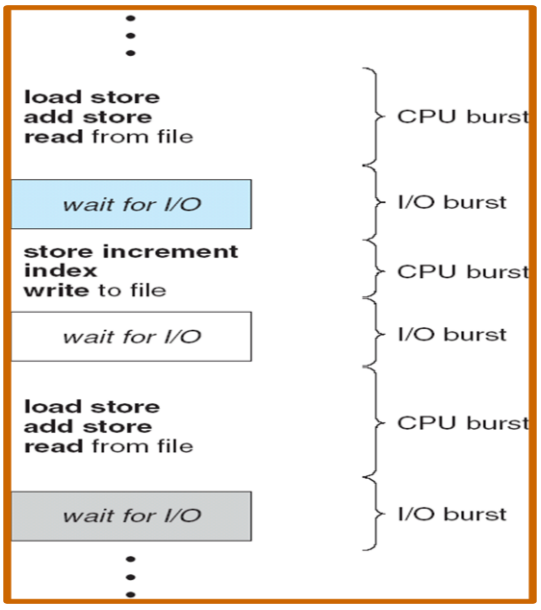


Figure 2.2: Diagram illustrating Alternating Sequence of CPU and I/O Burst

2.3 Scheduling

Scheduling is the method or actions by which threads, processes or data flows are given access to the system resources (e.g. process time, CPU, communication, bandwidth) which is usually done to balance a system and achieve best quality of service (Ernst *et al*, 2007). 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. The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously). The basic idea is to keep the CPU busy as much as possible by executing a user process until it must wait for an event, and then switch to another process. 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).

2.3.1 Schedulers

Scheduler is the module that moves jobs from queue to queue as shown in Fig 2.3 of the Queuing diagram of process scheduling.

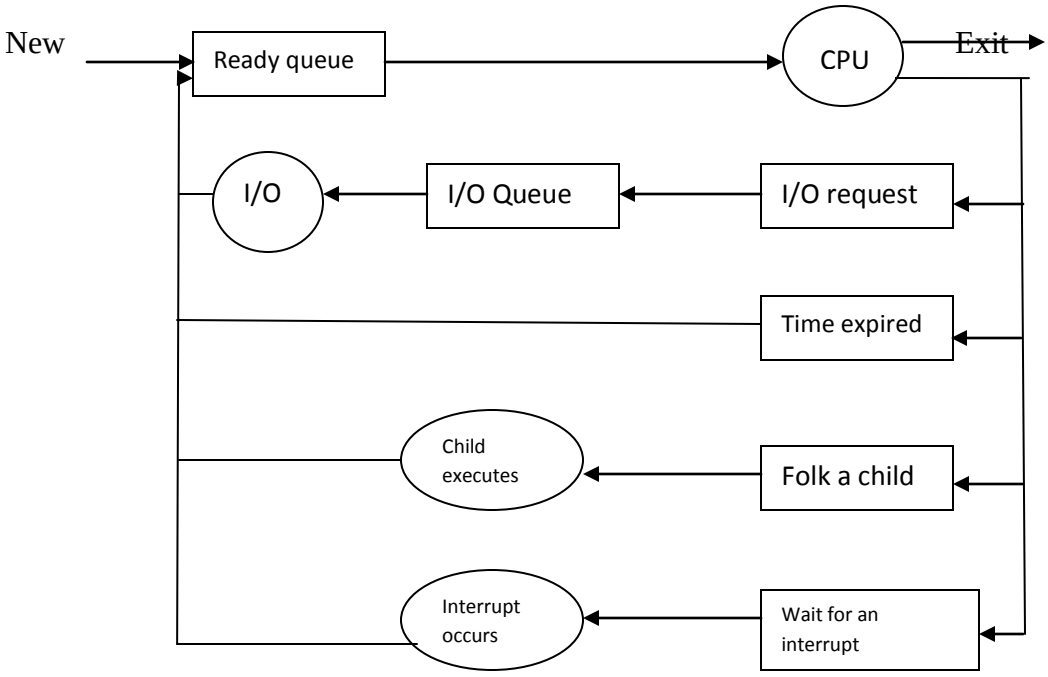


Figure 4.3 Queuing diagram of a process scheduling

Long term, Mid-term/medium-terms and short-term schedulers are the main types we will be looking at in this research. From the definition of scheduling and using Fig 2.3 to illustrate, schedulers are modulesthat choose and select the processes in and out of the CPU and orders processes to be admitted to the system.

2.3.2 Long-term Scheduler/admission Scheduler

From the concept of memory in operating system (Stallings, 2011), the Long-term scheduler decides which jobs are to be given and taken to the Ready queue; this scheduler is mainly concerned with the main memory. In scheduling concepts, the first and second phase of the process state diagram in Fig.2.1 will be analyzed. The Long-term scheduler also dictates to the system its degree of concurrency and how the processes will be split. Modern operating systems such as Windows have high priority to real-time process than others (Krishnan, 1995). The Long-term queue for this scheduling type exists in the hard disk or virtual memory (Stallings,

2001). Some existing examples of the Long-term scheduling are implemented in large scale system such as batch processing, computer clusters and super computers.

2.3.3 Mid/Medium-term Scheduler

Mid-term scheduler is best understood when related to the memory hierarchy concept in operating system, as such, the memory scale shows that from the main memory to the disk drives, the medium scheduler temporarily removes the processes from main memory to the secondary memory or vice versa as shown on Fig 2.4. This process is known as "swapping in and swapping out" also called "paging in and paging out" in some books (Silberschatz*et al.*, 2006) and (Prashant, 2008). The mid scheduler works in a way that it swaps out processes that have not been active or lower in priority measures, faulted processes, large acquiring memory process, request processes and later swaps them in when ready. Conclusively, the mid- term scheduler can be discussed as the middle module for swapping in and out of the memory. In modern operating systems, the mid-term scheduler performs the job of the long-term scheduler using the virtual address space technology of the memory.

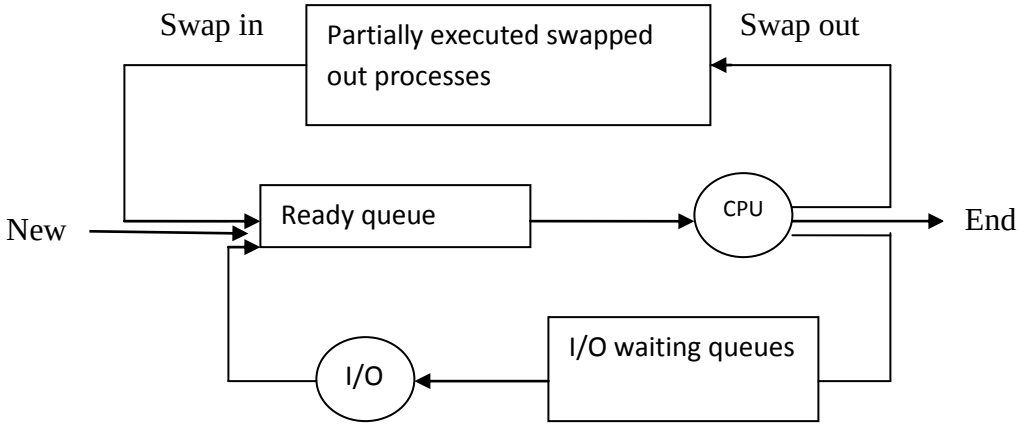


Figure 2.4: Mid-term scheduler

2.3.4 Short-term Scheduler/CPU scheduler

This is the scheduler that deals with the Ready Queue as seen in Fig 2.4. It chooses the processes to be executed and assigned to the CPU. Going by the process state diagram in Fig 2.1 and the Queuing diagram in Fig 2.3, the short-term scheduler decides more than the long-term scheduler. It works based on clock interrupt, I/O interrupts and system calls. It does the pre-emption (forcibly removing process from the CPU) depending on the algorithm, priority or time slice (Gerald *et al.*, 2004).

2.3.5 Dispatcher

In every operating system, scheduling is best treated alongside the dispatcher. The dispatcher is a module that gives privileges/control of the CPU to the process selected by the Short-term scheduler. The functions of the dispatcher include:

a. Context Switching
b. User mode Switching
c. Program location jumping

The time it usually takes to stop a process and start another running program is known as the *dispatcher latency*.

Scheduling as a concept of operating system migrates jobs into various algorithms which are used to distribute resources amongst processes. The common algorithms in operating systems include the First Come First Serve (FCFS), Shortest Job First (SJF), Longest Job First (LJF), Priority scheduling and the Round Robin (RR).

2.4 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 subsections below:

2.4.1 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.

Advantages

1. It is simple and has low overhead
2. The code for FCFS scheduling is simple to write and understand

Disadvantages

1. A long CPU-bound job may dominate the CPU and may force shorter jobs to wait for prolonged periods.
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.
3. Minimal average CPU utilization or average throughput (Oyetunji and Oluleye, 2009).

2.4.2 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 to 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 running 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. Preemptive SJF is sometimes referred to as shortest remaining time first scheduling.

Advantage

1. It gives the minimum average waiting time and minimum average turnaround time for a given set of processes (Oyetunji and Oluleye, 2009).

Disadvantages

1. It is difficult to know the length of the next CPU burst
2. Long running processes may starve, because the CPU has a steady supply of short processes(Suri and Sumit, 2012; Silberschatz*et al.,* 2006).

2.4.3 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. (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. Internally 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.

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.

Advantage

1. Good response for the highest priority processes.

Disadvantages

1. It suffers 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.

2.4.4 Round Robin Scheduling (RR)

Round Robin scheduling is designed for time-sharing systems. 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

equal to its 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 a given time quantum, and dispatches the process.

One of two things will then happen. The process may have a CPU burst of less than the 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 the time quantum, the timer goes off, and will interrupt 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 to the CPU for more than its time quantum in a row (unless it is the only runnable process). If a process's CPU burst exceeds its time quantum, that process will be preempted and put in the ready queue. The Round Robin scheduling algorithm is thus preemptive.

Advantages

1. Round-Robin is effective in a general-purpose, time-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.

Disadvantages

1. Care must be taken in choosing quantum value.

2. Throughput is low if time quantum is too small

2.5 Scheduling criteria

The various CPU scheduling algorithms have different properties and the choice of a particular algorithm may favor one class of processes over another. For selection of an algorithm for a particular situation, we must consider properties of various algorithms.

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:

1. Context Switch: A context switch is the 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, leads 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 and the goal, is to minimize it.
2. Throughput:Throughput is defined as the number of processes completed per unit time. Throughput is low in Round Robin scheduling implementation. Context switching and throughput are inversely proportional.
3. CPU Utilization: This is a measure of how 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 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 the 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 we can conclude that a good scheduling algorithm for real time and time sharing system must possess the following characteristics (Ajit*et al*., 2010):

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

2.6 Related Work

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 processor to the processes.

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 and to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently that the 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).

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 allocated to the processor for certain periods of time known as time quantum (q) or time slice. If a process is executed to the finish within a given time quantum, it 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 queue. The algorithm continues until all the processes are terminated, or the system is switched off.

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

willlead to the algorithm reverting to First Come First Serve scheduling algorithm. 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, response time and number of context switchesof all the processes.The importance of the problem has already raised the attention of researchers and study in this field still continues. Some more important works are listed below:

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 works by allocating the CPU to every process in Round Robin fashion with an initial time quantum (say k units). After completing the first cycle, it doubles the initial time quantum (2k units); selects the shortest process from the waiting queue and assigns the CPU to it for a period corresponding to its new time quantumand 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 be executed using the doubled time quantum.The time quantum is doubled after each cycle until all processes are executed and removed from the ready queue. This algorithm assumes that all processes arrive at the same 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 the least average waiting time, average turnaround time and number of context switches as compared to simple Round Robin. This in turn reduces the operating system overhead and dispatch latency. Also, it reduces the problem of starvation as the processes with minimum remaining CPU burst time are assigned higher priorities and are executed first in the

second round of the algorithm. The algorithm works by allocating CPU to every process in Round Robin fashion according to their given priorities for a given time quantum (say k units) on for only the first cycle. After completing the execution of the processes in the first cycle, 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 shows 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 corresponding to a given 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 the time quantum, the CPU is again allocated to the currently running process for the remaining CPU burst time. In this case, the 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 the 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.

Abdulrahim*et al.,* (2014) developed an algorithm that describes an improvement to the algorithm by Manish and AbdulKadir(2012)and also proved the experimental results of its performance over simple Round Robin scheduling algorithm and the algorithm by Manish and AbdulKadir(2012). The results show that the proposed algorithm gives better results in terms of minimizing average waiting time, average turnaround time and number of context switches in all cases of process categories than the simple Round Robin CPU scheduling algorithm and the algorithm by Manish and AbdulKadir(2012). This proposed algorithm works in a similar way as the algorithm by Manish and AbdulKadir(2012)but with some modification. It worksin three stages:

Stage 1: It picks the first process that arrives on the ready queue and allocates the CPU to it for a time interval equal to itstime quantum. After completion of process's time quantum, it checks the remaining CPU burst time of the currentlyrunning process. If the remaining CPU burst time of the currently running process is less or equal to the time quantum, theCPU is again allocated to the currently running process for remaining CPU burst time. In this case, the process will finishexecution and it will be removed from the ready queue. The scheduler then proceeds to the next shortest process in theready queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than the allocated time quantum,the process will be put at the tail of the ready queue.

Stage 2: The CPU scheduler will then select the next shortest process in the ready queue, and do the process in stage 1.

Stage 3: For the complete execution of all the processes, stage 1 and Stage 2 have to be repeated.

Behera*et al.,* (2010)developed an algorithm (A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis) and proved the

experimental results of its performance over simple Round Robin scheduling algorithm. This algorithm reduces the number of context switches, 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 into 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.

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 usesHighest Response Ratio (HRR) criteria for selecting processes from Ready Queue. They proved their experimental results by analyzing and comparing the algorithm with Behera*et al* (2010)in terms of reducing the number of context switches, average waiting time and average turnaround time. It does this by usingHighest 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. 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 the formula below. The algorithm takes dynamic time quantum into account.

$$ResponseRatio = \frac{waitingtime + bursttime}{bursttime} \qquad 2.1$$

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 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 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 assigning the CPU to it for a given time quantum. The difference shall be:

1. Behera*et al.*, (2012) combines simple RR with HRRN, while we adopt HRRN to replace SJN in determining the next process to be allocated the CPU in the algorithm by Abdulrahim*et al* (2014).
2. Behera*et al.*, (2012) uses the mean of the burst times in dynamically determining the time quantum while the user will input the time quantum before the execution of the processes (static provision) in our own case.

CHAPTER THREE

DESIGN OF THE IMPOROVED PROPOSED ROUND ROBIN CPU SCHEDULING ALGORITHM

This chapter presents the proposed Round Robin CPU scheduling algorithm and its Pseudo code describing how it works with the aid of an illustrative example; followed by the description of the various scheduling algorithms under study with the aid of illustrative example and finally italso states the system architecture.

3.1 The Proposed Improved Round Robin with Highest Response Ratio Next (IRRHRRN) Scheduling Algorithm

The proposed IRRHRRNCPU scheduling algorithm is animprovement of the Improved Round Robin (IRR) CPU scheduling algorithms (i.e. the algorithms by Manish and AbdulKadir (2012) and Abdulrahim*et al* (2014)).For a given time quantum say k, the algorithm calculates the Response Ratio of all processes available in the ready queue using equation 3.1

$$ResponseRatio = \frac{waitingtime + bursttime}{bursttime} \qquad 3.1$$

And then assigns the CPU to the process with the Highest Response Ratio Next (HRRN). After executing the process for itsallocated time quantum (k), it checks if the remaining burst time of the currently running process is less than or equal to k. If it is, then the currently running process will be reallocated to the CPU, finish its execution and will be removed from the ready queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than k, the process will be moved to the tail of the ready queue. It recalculates the Response Ratio of all

processes available in the ready queue taking into consideration, the newly arrived processes using equation 3.1

The CPU will be allocated to the processwith the Highest Response Ratio Next (HRRN) in the ready queue. After executing the process for its time quantum, it checks if the remaining burst time of the currently running process is less than or equal to the time quantum. If it is, then the currently running process will be reallocated to the CPU, finish its execution and will be removed from the ready queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than the time quantum, the process will be moved to the tail of the ready queue. These activities continue until no process is available in the ready queue.

3.1.1 The pseudo code of the proposed Improved Round Robin with Highest Response Ratio Next (IRRHRRN) Scheduling Algorithm

Step 1: Start

INPUT: Number of Processes (N), Burst Time (BT) of processes, Lower Bound (LB) andUpper Bound (UB) of Burst Time, Arrival Time (AT) of processes, Time quantum, Queue REQUEST, rate

OUTPUT: Number of Context Switches (NCS), Average Waiting Time (AWT), Average Turnaround Time (ATAT) and Average Response Time (ART)

Step 2: $NCS = 0;$

$$AWT = 0;$$

$ATAT = 0;$

$RT = 0;$

$BT = \text{uniform (LB, UB)}$;

$AT = Exponential\ ()$;

$Last\ Point = 0$

$$n = N;$$

Step 3: WHILE (REQUEST! = NULL)

Step 4: For $i = 1\ to\ n$//Make a ready queue

$$if\ AT_i \leq Last\ Point$$

$REQUEST \leftarrow Process_i$//Fill the ready queue according to Arrival Time

END if

END For

Step 5: For $i = 1\ to\ n$

$RR_i = \frac{UWT_i + BT_i}{BT_i}$ //calculate Response Ratio of each process

END For

$$Min = 0$$

Step 6: For$i = 1\ to\ n - 1$

If $RR_i > Min$//sort processes according to highest RR_i

$$Min = RR_i$$

$$RR_i = RR_{i+1}$$

$$RR_{i+1} = Min$$

END if

END For

Step 7: If $((BT_i < TQ)||(BT_i \leq 2 * TQ))$ //Assign CPU to the process with the highest RR_i

If $(exec_i == false)$ {

$$Last\ Point = Last\ Point + BT_i$$

$$BT_i = 0$$

$$TAT_i = Last\ Point - AT_i$$

$$WT_i = TAT_i - BT_i$$

$$RT_i = TAT_i - BT_i$$

$$exec_i = true$$

$$NCS_i = NCS_i + 1$$

}

Else {

$$Last\ Point = Last\ Point + BT_i$$

$$TAT_i = Last\ Point - AT_i$$

$$WT_i = TAT_i - BT_i$$

$$BT_i = 0$$

$$NCS_i = NCS_i + 1$$

}

Else {

If $(exec_i == false)$ {

$$BT_i = BT_i - TQ$$

$$RT_i = TQ$$

$$exec_i = true$$

$$NCS_i = NCS_i + 1$$

$$Last\ Point = Last\ Point + TQ$$

}

Else {

$$BT_i = BT_i - TQ$$

$$NCS_i = NCS_i + 1$$

$$Last\ Point = Last\ Point + TQ$$

}

}

END if

Step 8: For $i = 1\ to\ n$// Update REQUEST

$$if\ AT_i \leq Last\ Point$$

$REQUEST \leftarrow Process_i$//Fill the ready queue according to Arrival Time

END if

END For

Step 9: For process $i = 1\ ton$

$Update\ UWT_i$

END For

Step 10: END WHILE

Step 11: uniform (a, b) {

$$a + rand(0,1) * (b - a)$$

}

Step 12: $Exponential$ (lambda) {

$$return\ (lambda * e^{-lambda*mean})$$

}

Step 13: Calculate OUTPUT parameters

$$AWT = \frac{\sum_{i=1}^{n} WT_i}{n}$$

$$ATAT = \frac{\sum_{i=1}^{n} TAT_i}{n}$$

$$ART = \frac{\sum_{i=1}^{n} RT_i}{n}$$

$$NCS = \sum_{i=1}^{n} NCS_i$$

Step 14: END

3.2 How the Proposed Algorithm works

Following all steps of the proposed algorithm stated in section3.1.1, we will illustrate how it works with the aid of an example. Given 5 processes with their arrival and burst times shown in Table 3.1, the time quantum used is 10*ms*. The proposed algorithm works as follows

Table 3.1: Process Table

PROCESS ID	BURST TIME	ARRIVAL TIME
P1	27	0
P2	7	4
P3	30	10
P4	36	15
P5	18	17

As P1 arrive the system, the CPU will be allocated to it for the period equal to the time quantum (10*ms*). This is because it is the only process that isin the system. After execution, the remaining burst time of P1 is 17*ms,* which is greater than the time quantum. So, it will be preempted and the CPU will be assigned to the next process to be executed. To select the next process, the Highest Response Ratio (HRR) must be determined. At this point, only P2 and P3 have arrived in the system, so their respective Response Ratio using equation 3.1 above will be 1.86 and 1.

P2 has the HRR (that is, 1.86), so it will be executed for the period of its burst time (i.e. 7*ms*); this is because it is less than the time quantum (that is, 10*ms*) and will be removed from the system. After the execution of P2, all the remaining processes have arrived in the system and their individual Response Ratio will be calculatedusing equation 3.1. The Response Ratio of P1, P3, P4 and P5 will be 1.4, 1.2, 1.1 and 1 respectively. P1 has the HRR (that is, 1.4) and it will be executed for the period of its burst time (that is, 17*ms*); this is because after executing for a period equal to the time quantum (that is, 10*ms*), its remaining burst time (that is, 7*ms*) is less than the time quantum (that is, 10*ms*) and the CPU will be reallocated to it for a period of the remaining burst time (that is, 7*ms*) and then, it will be removed from the system. After the execution of P1, new Response Ratio will be calculated for each process using equation 3.1 above. The Response Ratio of P3, P4 and P5 will be 1.8, 1.5 and 1.9 respectively.P5 has the HRR (that is, 1.9) and it will be executed for the period of its burst time (that is, 18*ms*); this is

because after executing for a period equal to the time quantum (i.e. 10*ms*), its remaining burst time (that is, 8*ms*) is less than the time quantum (that is, 10*ms*) and the CPU will be reallocated to it for a period of the remaining burst time (that is, 8*ms*) and then, it will be removed from the system. After the execution of P5, new Response Ratio will be calculated for each process using equation 3.1. The Response Ratio of P3 and P4 will be 2.4 and 2.0 respectively.P3 has the HRR (that is, 2.4) and it will be executed for the period equal to the time quantum (10*ms*). This is because its remaining burst time is 20*ms* which is greater than the time quantum. So, it will be preempted and new RR will be calculated for each process. The new Response Ratio of P3 and P4 will be 1 and 2.3 respectively using equation 3.1.P4 has the HRR (that is, 2.3) and it will be executed for the period equal to the time quantum (10*ms*). This is because its remaining burst time is 26*ms* which is greater than the time quantum. So, it will be preempted and new RR will be calculated for each process. The new Response Ratio of P3 and P4 will be 1.5 and 1.0 respectively using equation 3.1 above.P3 has the HRR (that is, 1.5) and it will be executed for the period of its burst time (that is, 20*ms*); this is because after executing for the given time quantum (that is, 10*ms*), its remaining burst time (that is, 10*ms*) is equal to the time quantum (that is, 10*ms*) and the CPU will be reallocated for a period of the remaining burst time (that is, 10*ms*) and then it will be removed from the system. At this point, only P4 is left in the system and it will be executed for the period of its burst time (that is, 26*ms*).

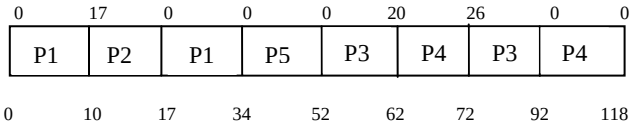


Figure 3.1: Gantt chart representation of the illustrated example for the Proposed Algorithm

3.3 Illustrative Examples

To demonstrate the previous considerations, we will consider this example, in which each process with its burst and arrival time as shown in Table 3.2, where the time quantum used in RR, IRR, AAIRR and IRRHRRN is 10*ms*while the time quantum for RRHRRN is determined by taking the mean of the burst time.

Table 3.2: Process Table 2

PR_ID	AT	BT
P1	0	1
P2	1	6
P3	2	3
P4	5	30
P5	5	32
P6	7	21
P7	9	2
P8	10	13
P9	11	2
P10	15	35

For evaluation purpose, the formula of Waiting Time (WT) (that is, (3.2)) shall be used in calculating the Average Waiting Time (AWT)(that is, (3.3)) for each scheduling algorithm.

$$Waiting\ Time = Time\ first\ scheduled - Arrival\ Time \quad 3.2$$

$$Average\ Waiting\ Time = \frac{Sum\ of\ all\ processes\ Waiting\ Time}{Number\ of\ processes} \quad 3.3$$

The Turnaround Time (TAT) (that is, (3.4)) shall also be used in calculating the Average Turnaround Time (ATAT) (that is, (3.5)) for each scheduling algorithm.

$$Turnaround\ Time = Time\ of\ process\ completion - Arrival\ Time \quad 3.4$$

$$Average\ Turnaround\ Time = \frac{Sum\ of\ all\ processes\ Turnaround\ Time}{Number\ of\ processes} \quad 3.5$$

And the Response Time (RT) (that is, 3.6) shall also be used in calculating the Average Response Time (ART) (that is, 3.7) for each scheduling algorithm.

$$ResponseTime = Time\ of\ process's\ first\ response - Arrival\ Time \quad 3.6$$

$$Average\ Response\ Time = \frac{Sum\ of\ all\ processes\ Response\ Time}{Number\ of\ processes} \quad 3.7$$

3.3.1 Round Robin (RR)

Figure 3.2 shows the Gantt chart of Round Robin CPU scheduling algorithm

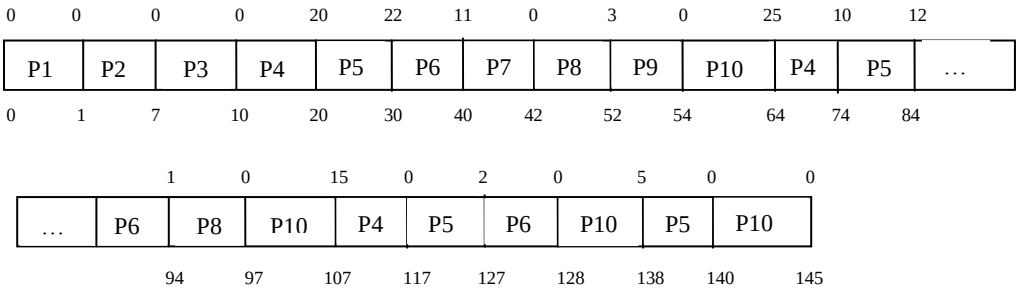


Figure 3.2: Gantt chart representation of RR

Waiting Time

P1: (0-0) = 0, P2: (1-1) =0, P3: (7-2) = 5, P4: ((10+44+33)-5) = 146, P5: ((20+44+33+11)-5) = 108, P6: ((30+44+33)-7) = 100, P7: (40-9) = 31, P8: ((42+42)-10) = 74, P9: (52-11) = 41, P10: ((54+33+21+2)-15) = 95.

Average Waiting Time

$$AWT = \frac{(0 + 0 + 5 + 82 + 103 + 100 + 31 + 74 + 41 + 95)}{10} = \frac{531}{10} = 53.1$$

Turnaround Time

P1: (1-0) = 1, P2: (7-1) =6, P3: (10-2) = 8, P4: (117-5) =112, P5: (140-5) = 135, P6: (128-7) = 121, P7: (42-9) = 33, P8: (97-10) = 87, P9: (54-2) = 52, P10: (145-15) = 130.

Average Turnaround Time

$$ATAT = \frac{1 + 6 + 8 + 112 + 135 + 121 + 33 + 87 + 52 + 130}{10} = \frac{676}{10} = 67.6$$

Response Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (10-5) = 5, P5: (20-5) = 15, P6: (30-7) = 23, P7: (40-9) = 31, P8: (42-10) = 32, P9: (52-11) = 41, and P10: (54-15) = 39.

Average Response Time

$$ART = \frac{(0 + 0 + 5 + 5 + 15 + 23 + 31 + 32 + 41 + 39)}{10} = \frac{191}{10} = 19.1$$

Number of Context Switches = 20

3.3.2 First Come First Serve (FCFS)

Figure 3.3 shows the Gantt chart of First Come First Serve CPU scheduling algorithm

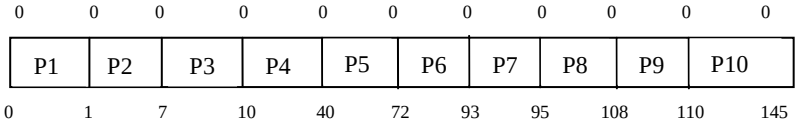


Figure 3.3: Gantt chart representation of FCFS

Waiting Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (10-5) = 5, P5: (40-5) = 35, P6: (72-7) = 65, P7: (93-9) = 84, P8: (95-10) = 85, P9: (108-11) = 97, and P10: (110-15) = 95.

Average Waiting Time

$$AWT = \frac{(0 + 0 + 5 + 5 + 35 + 65 + 84 + 85 + 87 + 95)}{10} = \frac{471}{10} = 47.1$$

Turnaround Time

P1: (1-0) = 1, P2: (7-1) =6, P3: (10-2) = 8, P4: (40-5) =35, P5: (72-5) = 67, P6: (93-7) = 86, P7: (95-3) = 93, P8: (108-10) = 98, P9: (110-10) = 100, P10: (145-15)= 130.

Average Turnaround Time

$$ATAT = \frac{1 + 6 + 8 + 35 + 67 + 86 + 93 + 98 + 100 + 130}{10} = \frac{616}{10} = 61.6$$

Response Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (10-5) = 5, P5: (40-5) = 35, P6: (72-7) = 65, P7: (93-9) = 84, P8: (95-10) = 85, P9: (108-11) = 97, and P10: (110-15) = 95.

Average Response Time

$$ART = \frac{(0 + 0 + 5 + 5 + 35 + 65 + 84 + 85 + 87 + 95)}{10} = \frac{471}{10} = 47.1$$

Number of Context Switches = 0

3.3.3 Improved Round Robin (IRR)

Figure 3.4 shows the Gantt chart of Improved Round Robin CPU scheduling algorithm

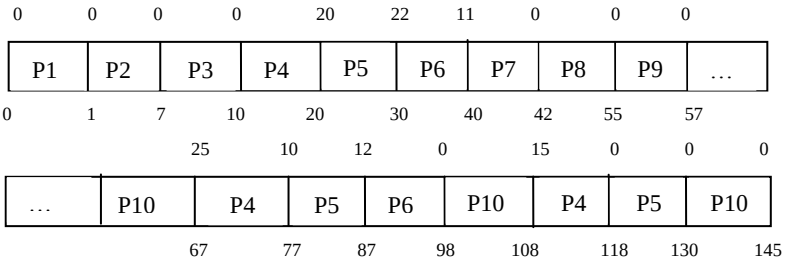


Figure 3.4: Gantt chart representation of IRR

Waiting Time

P1: (0-0) = 0, P2: (1-1) =0, P3: (7-2) = 5, P4: ((10+47+31)-5) = 83, P5: ((20+47+31)-5) = 93, P6: ((30+47)-7)=70, P7: (40-9) = 31, P8: ((48)-10) = 38, P9: (55-11) = 44, P10: ((57+31+22)-15) = 95.

Average Waiting Time

$$AWT = \frac{(0 + 0 + 5 + 83 + 93 + 70 + 31 + 38 + 44 + 95)}{10} = \frac{453}{10} = 45.3$$

Turnaround Time

P1: (1-0) = 1, P2: (7-1) =6, P3: (10-2) = 8, P4: (118-5) =113, P5: (130-5) = 125, P6: (98-7) = 91, P7: (42-9) = 33, P8: (55-10) = 45, P9: (57-2) = 55, P10: (145-15) = 130.

Average Turnaround Time

$$ATAT = \frac{1 + 6 + 8 + 113 + 125 + 91 + 33 + 45 + 55 + 130}{10} = \frac{598}{10} = 59.8$$

Response Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (10-5) = 5, P5: (20-5) = 15, P6: (30-7) = 23, P7: (40-9) = 31, P8: (42-10) = 32, P9: (55-11) = 44, and P10: (57-15) = 42.

Average Response Time

$$ART = \frac{(0+0+5+5+15+23+31+32+44+42)}{10} = \frac{197}{10} = 19.7$$

Number of Context Switches = 16

3.3.4 An Additional Improvement in Round Robin (AAIRR)

Figure 3.5 shows the Gantt chart of An Additional Improvement in Round Robin CPU scheduling algorithm

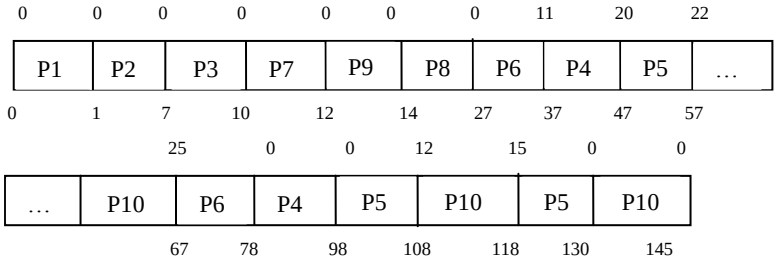


Figure 3.5: Gantt chart representation of AAIRR

Waiting Time

P1: (0-0) = 0, P2: (1-1) =0, P3: (7-2) = 5, P4: ((37+31)-5) = 68, P5: ((47+41+10)-5) = 93, P6: ((27+30)-7)=50, P7: (10-9) = 1, P8: ((14)-10) = 4, P9: (12-11) = 1, P10: ((57+41+2)-15) = 85.

Average Waiting Time

$$AWT = \frac{(0+0+5+63+93+50+1+4+1+85)}{10} = \frac{302}{10} = 30.2$$

Turnaround Time

P1: (1-0) = 1, P2: (7-1) =6, P3: (10-2) = 8, P4: (98-5) =93, P5: (130-5) = 125, P6: (78-7) = 71, P7: (12-9) = 3, P8: (27-10) = 17, P9: (14-11) = 3, P10: (145-15) = 130.

Average Turnaround Time

$$ATAT = \frac{1+6+8+93+125+71+3+17+3+130}{10} = \frac{447}{10} = 44.7$$

Response Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (37-5) = 32, P5: (47-5) = 42, P6: (27-7) = 20, P7: (10-9) = 1, P8: (14-10) = 4, P9: (12-11) = 1, and P10: (57-15) = 42.

Average Response Time

$$ART = \frac{(0 + 0 + 5 + 32 + 42 + 20 + 1 + 4 + 1 + 42)}{10} = \frac{147}{10} = 14.7$$

Number of Context Switches = 15

3.3.5 Round Robin with Highest Response Ratio Next (RRHRRN) Scheduling Algorithm

Figure 3.6 shows the Gantt chart of RRHRRN CPU scheduling algorithm

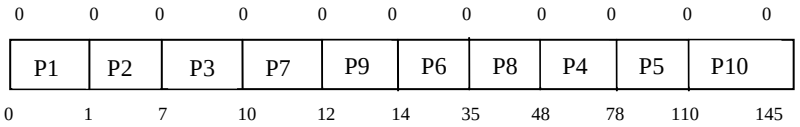


Figure 3.6: Gantt chart representation of RRHRRN

Waiting Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (48-5) = 43, P5: (78-5) = 73, P6: (14-7) = 14, P7: (10-9) = 1, P8: (35-10) = 25, P9: (12-11) = 1, and P10: (110-15) = 95.

Average Waiting Time

$$AWT = \frac{(0 + 0 + 5 + 43 + 73 + 7 + 1 + 25 + 1 + 95)}{10} = \frac{250}{10} = 25.0$$

Turnaround Time

P1: (1-0) = 1, P2: (7-1) =6, P3: (10-2) = 8, P4: (78-5) =73, P5: (110-5) = 105, P6: (35-7) = 28, P7: (12-9) = 3, P8: (48-10) = 38, P9: (14-11) = 3, P10: (145-15)= 130.

Average Turnaround Time

$$ATAT = \frac{1 + 6 + 8 + 73 + 105 + 28 + 3 + 38 + 3 + 130}{10} = \frac{395}{10} = 39.5$$

Response Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (48-5) = 43, P5: (78-5) = 73, P6: (14-7) = 14, P7: (10-9) = 1, P8: (35-10) = 25, P9: (12-11) = 1, and P10: (110-15) = 95.

Average Response Time

$$ART = \frac{(0 + 0 + 5 + 43 + 73 + 7 + 1 + 25 + 1 + 95)}{10} = \frac{250}{10} = 25.0$$

Number of Context Switches = 9

3.3.6 Improved Round Robin With Highest Response Ratio Next (IRRHRRN) CPU Scheduling Algorithm

Figure 3.7 shows the Gantt chart of the proposed Improved Round Robin with Highest Response Ratio Next (IRRHRRN) CPU Scheduling Algorithm

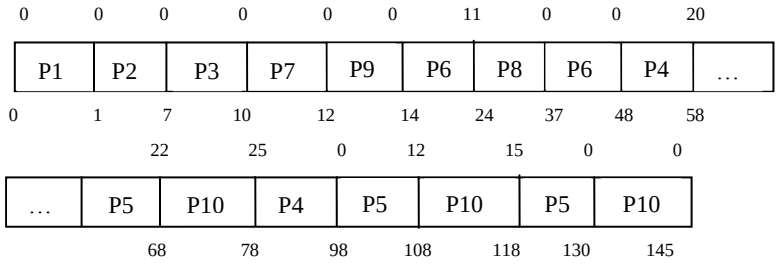


Figure 3.7: Gantt chart representation of IRRHRRN

Waiting Time

P1: (0-0) = 0, P2: (1-1) =0, P3: (7-2) = 5, P4: ((48+20)-5) = 63, P5: ((58+30+10)-5) = 93, P6: ((14+13)-7)=20, P7: (10-9) = 1, P8: ((24)-10) = 14, P9: (12-11) = 1, P10: ((68+30+12)-15) = 85.

Average Waiting Time

$$AWT = \frac{(0 + 0 + 5 + 63 + 93 + 20 + 1 + 14 + 1 + 85)}{10} = \frac{292}{10} = 29.2$$

Turnaround Time

P1: (1-0) = 1, P2: (7-1) =6, P3: (10-2) = 8, P4: (98-5) =93, P5: (130-5) = 125, P6: (48-7) = 41, P7: (12-9) = 3, P8: (37-10) = 27, P9: (14-11) = 3, P10: (145-15) = 130.

Average Turnaround Time

$$ATAT = \frac{1 + 6 + 8 + 93 + 125 + 41 + 3 + 27 + 3 + 130}{10} = \frac{437}{10} = 43.7$$

Response Time

P1: (0-0) = 0, P2: (1-1) = 0, P3: (7-2) = 5, P4: (48-5) = 43, P5: (58-5) = 53, P6: (14-7) = 7, P7: (10-9) = 1, P8: (24-10) = 14, P9: (12-11) = 1, and P10: (68-15) = 53.

Average Response Time

$$ART = \frac{(0 + 0 + 5 + 43 + 53 + 7 + 1 + 14 + 1 + 53)}{10} = \frac{177}{10} = 17.7$$

Number of Context Switches = 15

Table 3.3: Comparative Table

Algorithms	AWT	ATAT	ART	NCS	TQ
RR	53.1	67.6	19.1	20	10
FCFS	47.1	61.8	47.1		
IRR	45.3	59.8	19.7	16	10
AAIRR	31.2	45.7	14.7	15	10
RRHRRN	25.0	39.5	25.0	9	14.5
IRRHRRN	29.2	43.7	17.7	15	10

Table 3.3 shows the comparative results of the algorithms under study. Though it appears RRHRRN is better in terms of AWT and ATAT and NCS, it is not because it selects its time quantum dynamically while others select theirs statically (14.5ms for RRHRRN as against 10ms for other algorithms). This makes its comparison with other algorithms needless as there is no fair basis under which they can be compared. We shall ignore RRHRRN in our future discussions; we just included it because it is closely related to our work.

The proposed IRRHRRN has the minimal AWT and ATAT while FCFS has the minimal NCS. But in the Round Robin category, AAIRR and the proposed IRRHRRN have the minimal NCS while RR has the minimal ART.

3.4 System Architecture

The architecture of the system we developed is depicted in Figure 3.8. Let us now discuss how it works.

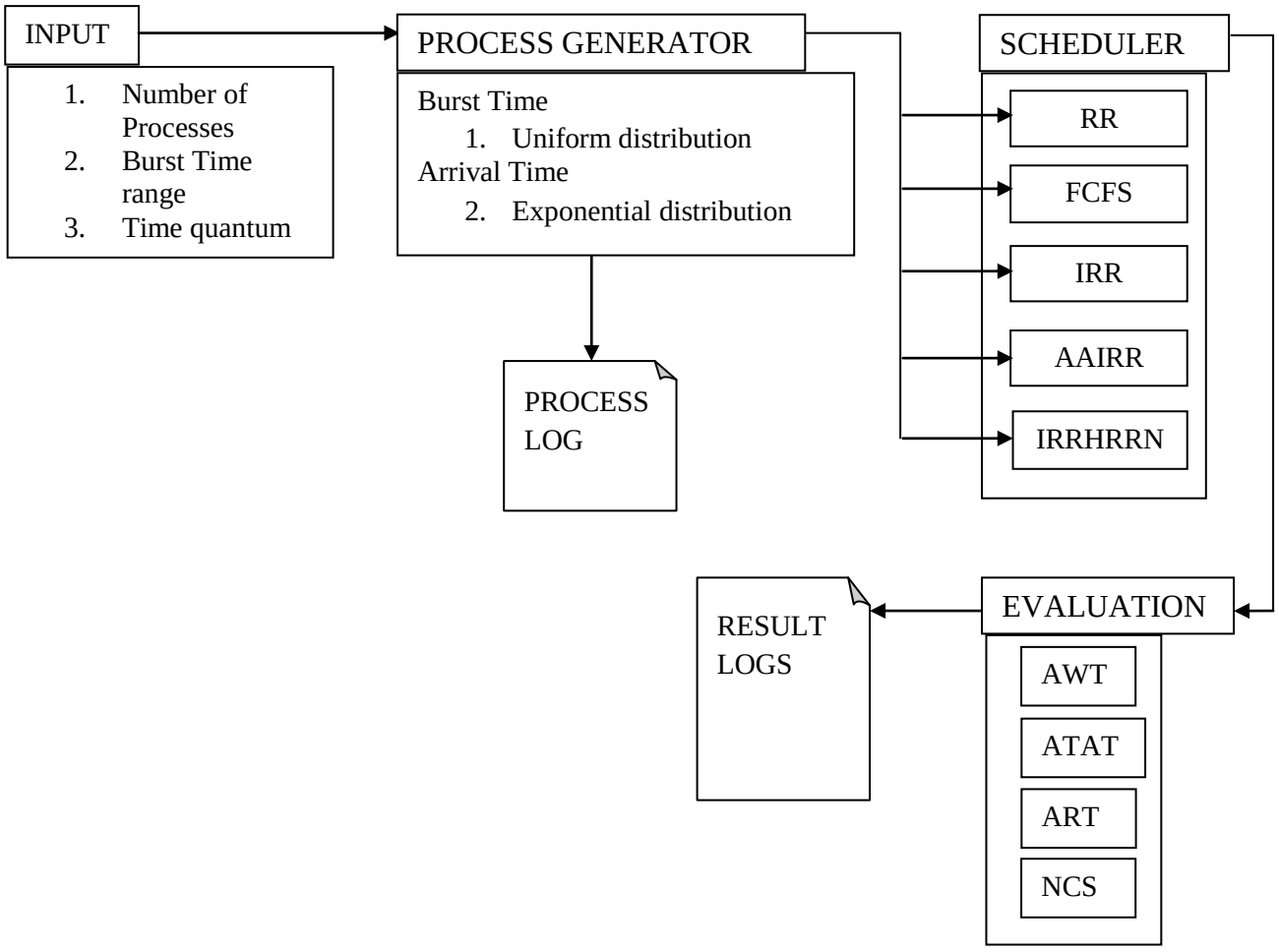


Figure 3.8: The System Architecture

The system takes in as input the number of processes to be considered, the range (lower and upper bound) of the burst times and the time quantum to be used by the scheduling algorithms. The burst times will be generated using the Uniform statistical distribution while the Exponentialstatistical distribution will be used for generating arrival times. The generated processes will be stored in the process log and will be arranged in ready queue according to their arrival times. The CPU will be allocated to the processes by using each of the scheduling algorithms in the scheduler and the performance criteria of each of the algorithms will be evaluated and stored in the criteria log where the values of each of the performance criteria will be used in assessing the scheduling algorithms.

CHAPTER FOUR

IMPLEMENTATION OF THE PROPOSED IMPROVED ROUND ROBIN WITH HIGHEST RESPONSE RATIO CPU SCHEDULING ALGORITHM

This chapter presents the implementation of the Proposed Improved Round Robin with Highest Response Ratio CPU scheduling algorithm.It starts with the assumptions adopted in the system design. It also states the system requirementsand finally, the system implementation follows; which is tested using processes that vary between 5 and 2000. The results of this implementation are analyzed

4.1 Assumption

We were interested in studying four performance criteria: average waiting time, average turnaround time, average response time and number of context switches. We simulated Round Robin (RR), First Come First Serve (FCFS), Improved Round Robin (IRR), An Additional Improvement in Round Robin and the proposed Improved Round Robin with Highest Response Ratio Next (IRRHRRN) CPU scheduling algorithms 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.

4.2 System Requirements

4.2.1 Experimental setup

Hardware

- Compaq Presario laptop with a AMD Sempron (tm) M120 processor running at 2.10GHz
- 2.0 GB of RAM and
- 150GB of hard disk

Software

- Window 7 Premium operating system
- NetBeans IDE 6.7.1 version and JDK1.7

We built a process generator routine 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 an Exponential random process. Hence, the inter-arrival times are exponentially 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 rate of arrival of the distribution function.

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:

$$f(x) = \lambda e^{-\lambda x} \qquad 4.1$$

Its mean and variance is given by:

$$E(X) = \frac{1}{\lambda} and\ V(X) = \frac{1}{\lambda^2} \qquad 4.2$$

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

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:

$$f(x) = \frac{1}{b-a} \tag{4.3}$$

Where $a \leq x \leq b$

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

$$F(x) = \frac{x-a}{b-a} = R \tag{4.4}$$

Or

$$x = a + (b-a)R \tag{4.5}$$

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

4.1 Results Discussion

Figure 4.1 shows the system interface through which the user can effectively communicate with the system. It takes 2000 processes as input with burst times that are exponentially distributed

ranging between 1 and 50*ms*, with mean of 25.5*ms*, parameter rate of 0.039 per *ms*, standard deviation of 14.15*ms* and the time quantum of 10*ms*.

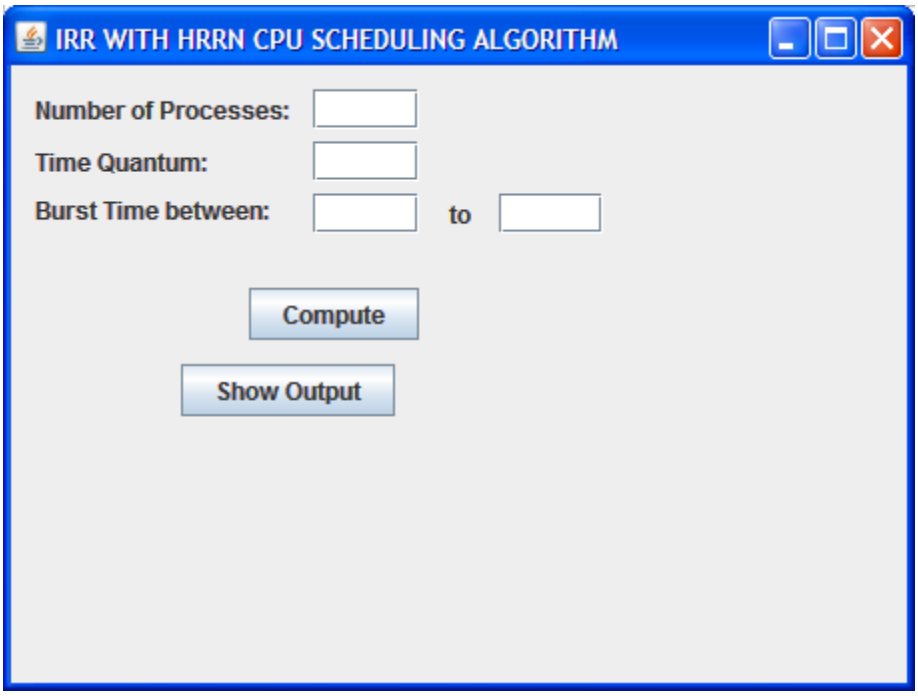


Figure 4.1: System Interface with some process execution in progress

Table 4.1 shows a snapshot of some of the 2000 processes (i.e. the ID, Burst time (*ms*) and Arrival time (*ms*)) used in computation.

Table 4.1: Process identity

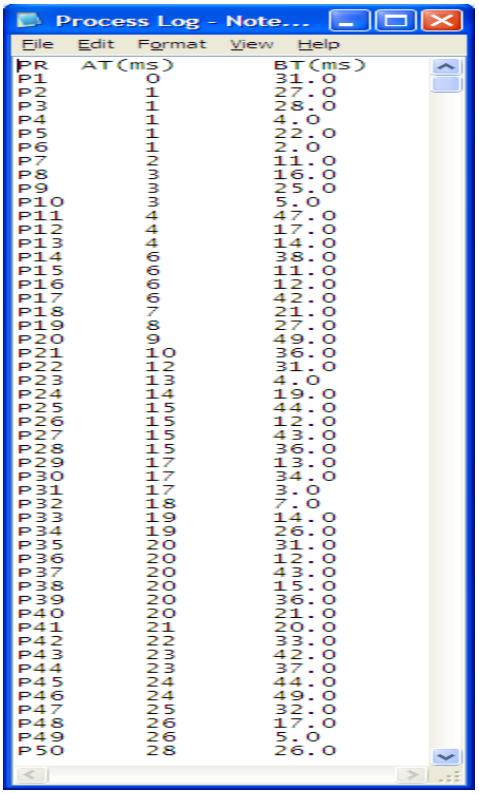


Table4.2 presents some of the results of Average Waiting Time for the different times of runs and Figure 4.2shows the overall graphical result of the Average Waiting Time for all cases of values taken. It was observed from the graph that our algorithm(that is, IRRHRRN) has the best performance, and AAIRR came second,followed byFCFS, IRR andRR respectively.

Table 4.2: Some of the Results of Average Waiting Times

Average Waiting Time - Notepad

File Edit Format View Help

PR	RR	FCFS	IRR	AAIRR	IRRHRRN
5	0.07	0.05	0.058	0.046	0.05
10	0.105	0.09	0.083	0.061	0.063
15	0.177	0.14	0.132	0.105	0.104
20	0.269	0.19	0.201	0.171	0.161
25	0.353	0.25	0.269	0.23	0.213
30	0.45	0.31	0.342	0.295	0.274
35	0.5	0.37	0.384	0.32	0.298
40	0.59	0.43	0.449	0.377	0.351
45	0.708	0.49	0.548	0.471	0.434
50	0.792	0.55	0.616	0.526	0.484
55	0.845	0.62	0.658	0.55	0.514
60	0.957	0.68	0.747	0.634	0.59
65	1.027	0.74	0.801	0.672	0.629
70	1.105	0.8	0.87	0.732	0.683
75	1.172	0.86	0.929	0.778	0.722
80	1.235	0.92	0.984	0.817	0.759
85	1.311	0.98	1.044	0.864	0.801
90	1.366	1.04	1.09	0.895	0.834
95	1.427	1.1	1.138	0.923	0.863
100	1.522	1.16	1.217	0.997	0.93
110	1.669	1.27	1.333	1.09	1.019
120	1.831	1.39	1.465	1.206	1.123
130	2.028	1.5	1.627	1.351	1.254
140	2.203	1.63	1.764	1.463	1.364
150	2.367	1.75	1.9	1.573	1.465
160	2.522	1.87	2.032	1.684	1.569
170	2.706	1.99	2.174	1.803	1.681
180	2.776	2.11	2.243	1.833	1.716
190	2.9	2.22	2.337	1.894	1.779
200	3.046	2.34	2.459	1.996	1.869
220	3.343	2.57	2.703	2.193	2.055
240	3.695	2.8	3.0	2.446	2.284
260	3.986	3.03	3.244	2.642	2.465
280	4.246	3.27	3.466	2.812	2.618
300	4.604	3.5	3.755	3.071	2.852
320	4.885	3.73	3.991	3.253	3.019
340	5.168	3.97	4.22	3.43	3.181
360	5.475	4.2	4.479	3.64	3.373
380	5.735	4.43	4.682	3.788	3.523
400	6.045	4.66	4.933	3.993	3.705
425	6.448	4.95	5.263	4.268	3.962
450	6.753	5.24	5.513	4.44	4.124
475	7.209	5.52	5.888	4.776	4.426
500	7.549	5.81	6.168	4.98	4.619
550	8.262	6.39	6.772	5.463	5.057
600	9.06	6.97	7.418	6.007	5.55
650	9.959	7.56	8.146	6.649	6.132
700	10.574	8.15	8.66	6.994	6.472
750	11.442	8.73	9.368	7.597	7.022
800	12.231	9.33	10.003	8.115	7.504

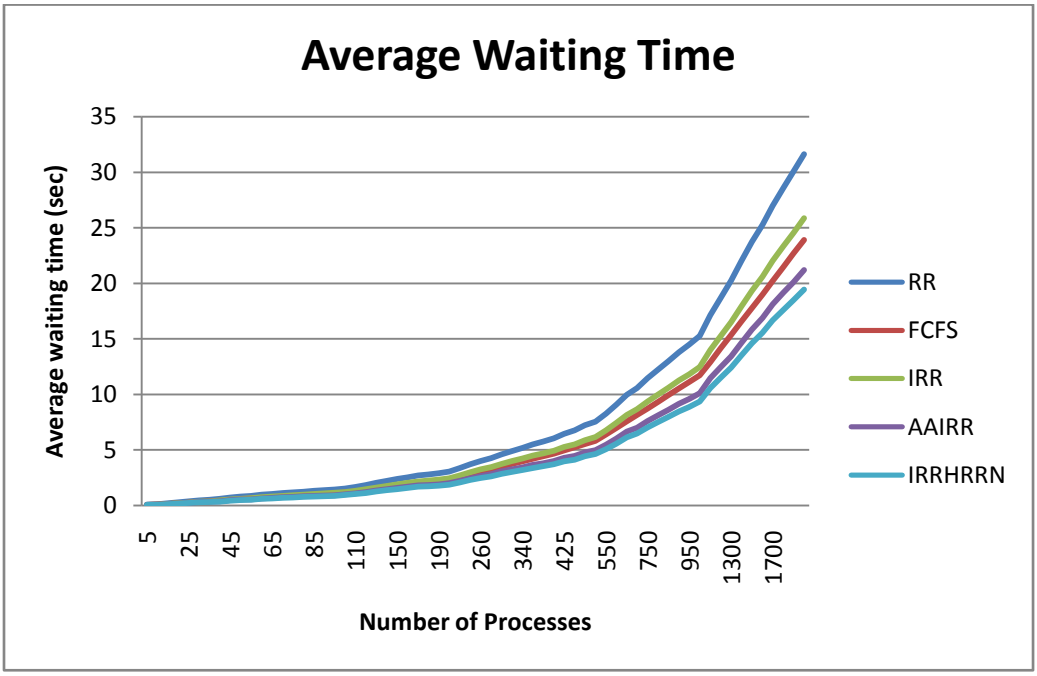
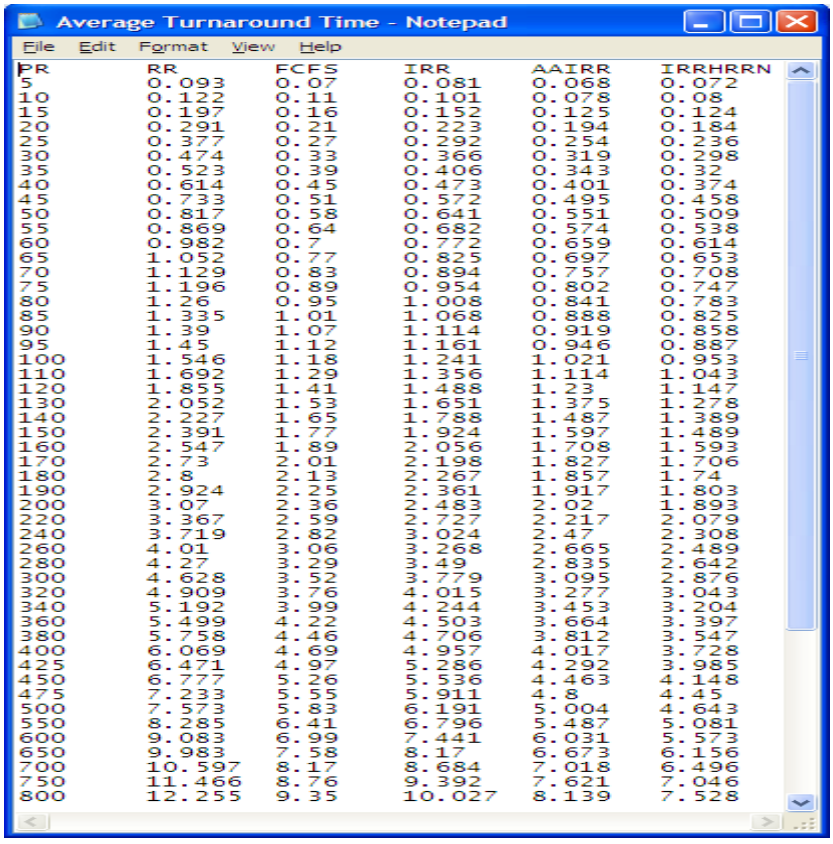


Figure 4.2: Graph of Average Waiting Time

Table4.3 presents some of the results of Average Turnaround Time for the different times of runs and Figure 4.3shows the overall graphical result of the Average Turnaround Time for all cases of values taken. It was observed from the graph that IRRHRRN has the best performance, and AAIRR came second,followed byFCFS, IRR andRR respectively.

Table 4.3: Some of the Results of Average Turnaround Times



PR	RR	FCFS	IRR	AAIRR	IRRHRRN
5	0.093	0.07	0.081	0.068	0.072
10	0.122	0.11	0.101	0.078	0.08
15	0.197	0.16	0.152	0.125	0.124
20	0.291	0.21	0.223	0.194	0.184
25	0.377	0.27	0.292	0.254	0.236
30	0.474	0.33	0.366	0.319	0.298
35	0.523	0.39	0.406	0.343	0.32
40	0.614	0.45	0.473	0.401	0.374
45	0.733	0.51	0.572	0.495	0.458
50	0.817	0.58	0.641	0.551	0.509
55	0.869	0.64	0.682	0.574	0.538
60	0.982	0.7	0.772	0.659	0.614
65	1.052	0.77	0.825	0.697	0.653
70	1.129	0.83	0.894	0.757	0.708
75	1.196	0.89	0.954	0.802	0.747
80	1.26	0.95	1.008	0.841	0.783
85	1.335	1.01	1.068	0.888	0.825
90	1.39	1.07	1.114	0.919	0.858
95	1.45	1.12	1.161	0.946	0.887
100	1.546	1.18	1.241	1.021	0.953
110	1.692	1.29	1.356	1.114	1.043
120	1.855	1.41	1.488	1.23	1.147
130	2.052	1.53	1.651	1.375	1.278
140	2.227	1.65	1.788	1.487	1.389
150	2.391	1.77	1.924	1.597	1.489
160	2.547	1.89	2.056	1.708	1.593
170	2.73	2.01	2.198	1.827	1.706
180	2.8	2.13	2.267	1.857	1.74
190	2.924	2.25	2.361	1.917	1.803
200	3.07	2.36	2.483	2.02	1.893
220	3.367	2.59	2.727	2.217	2.079
240	3.719	2.82	3.024	2.47	2.308
260	4.01	3.06	3.268	2.665	2.489
280	4.27	3.29	3.49	2.835	2.642
300	4.628	3.52	3.779	3.095	2.876
320	4.909	3.76	4.015	3.277	3.043
340	5.192	3.99	4.244	3.453	3.204
360	5.499	4.22	4.503	3.664	3.397
380	5.758	4.46	4.706	3.812	3.547
400	6.069	4.69	4.957	4.017	3.728
425	6.471	4.97	5.286	4.292	3.985
450	6.777	5.26	5.536	4.463	4.148
475	7.233	5.55	5.911	4.8	4.45
500	7.573	5.83	6.191	5.004	4.643
550	8.285	6.41	6.796	5.487	5.081
600	9.083	6.99	7.441	6.031	5.573
650	9.983	7.58	8.17	6.673	6.156
700	10.597	8.17	8.684	7.018	6.496
750	11.466	8.76	9.392	7.621	7.046
800	12.255	9.35	10.027	8.139	7.528

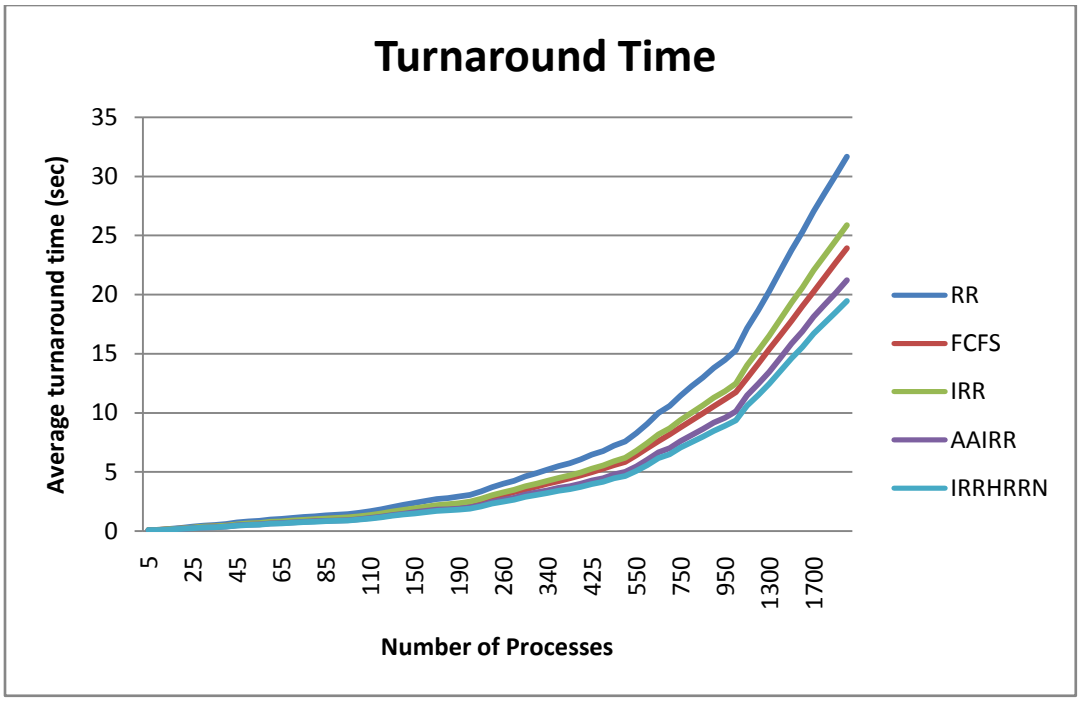


Figure 4.3 Graph of Average Turnaround Time

Table4.4 presents some of the results of Average Response Time for the different times of runs and Figure 4.4shows the overall graphical result of the Average Response Time for all cases of values taken. It was observed from the graph that this algorithm (that is, IRRHRRN) has the best performance, andRR came second,followed by IRR, AAIRR and FCFS respectively.

Table 4.4: Some of the Results of Average Response Times

Average Response Time - Notepad

File Edit Format View Help

PR	RR	FCFS	IRR	AAIRR	IRRHRRN
5	0.018	0.05	0.018	0.016	0.016
10	0.053	0.09	0.056	0.034	0.021
15	0.056	0.14	0.062	0.067	0.043
20	0.096	0.19	0.108	0.104	0.072
25	0.1	0.25	0.112	0.115	0.091
30	0.14	0.31	0.157	0.157	0.111
35	0.142	0.37	0.161	0.167	0.112
40	0.182	0.43	0.206	0.215	0.131
45	0.186	0.49	0.21	0.277	0.163
50	0.227	0.55	0.255	0.269	0.178
55	0.23	0.62	0.26	0.3	0.182
60	0.271	0.68	0.306	0.386	0.208
65	0.275	0.74	0.311	0.376	0.218
70	0.316	0.8	0.359	0.393	0.235
75	0.319	0.86	0.363	0.396	0.246
80	0.36	0.92	0.408	0.42	0.252
85	0.363	0.98	0.413	0.438	0.265
90	0.403	1.04	0.458	0.489	0.27
95	0.407	1.1	0.462	0.497	0.276
100	0.447	1.16	0.507	0.587	0.297
110	0.473	1.27	0.536	0.571	0.322
120	0.535	1.39	0.606	0.619	0.364
130	0.56	1.5	0.634	0.738	0.41
140	0.623	1.63	0.704	0.817	0.44
150	0.648	1.75	0.734	0.831	0.471
160	0.711	1.87	0.804	0.837	0.503
170	0.737	1.99	0.833	1.034	0.544
180	0.8	2.11	0.903	0.966	0.549
190	0.825	2.22	0.931	1.035	0.561
200	0.887	2.34	1.0	1.055	0.593
220	0.955	2.57	1.077	1.144	0.654
240	1.059	2.8	1.193	1.251	0.737
260	1.127	3.03	1.268	1.357	0.792
280	1.23	3.27	1.383	1.434	0.837
300	1.297	3.5	1.456	1.563	0.92
320	1.4	3.73	1.569	1.643	0.962
340	1.467	3.97	1.642	1.749	1.014
360	1.57	4.2	1.755	1.875	1.075
380	1.637	4.43	1.828	1.975	1.114
400	1.739	4.66	1.941	2.076	1.169
425	1.827	4.95	2.038	2.196	1.251
450	1.951	5.24	2.175	2.323	1.298
475	2.039	5.52	2.272	2.478	1.407
500	2.164	5.81	2.408	2.595	1.463
550	2.356	6.39	2.619	2.798	1.619
600	2.585	6.97	2.871	3.069	1.789
650	2.78	7.56	3.087	3.372	1.99
700	3.011	8.15	3.342	3.583	2.076
750	3.206	8.73	3.557	3.893	2.26
800	3.437	9.33	3.813	4.151	2.418

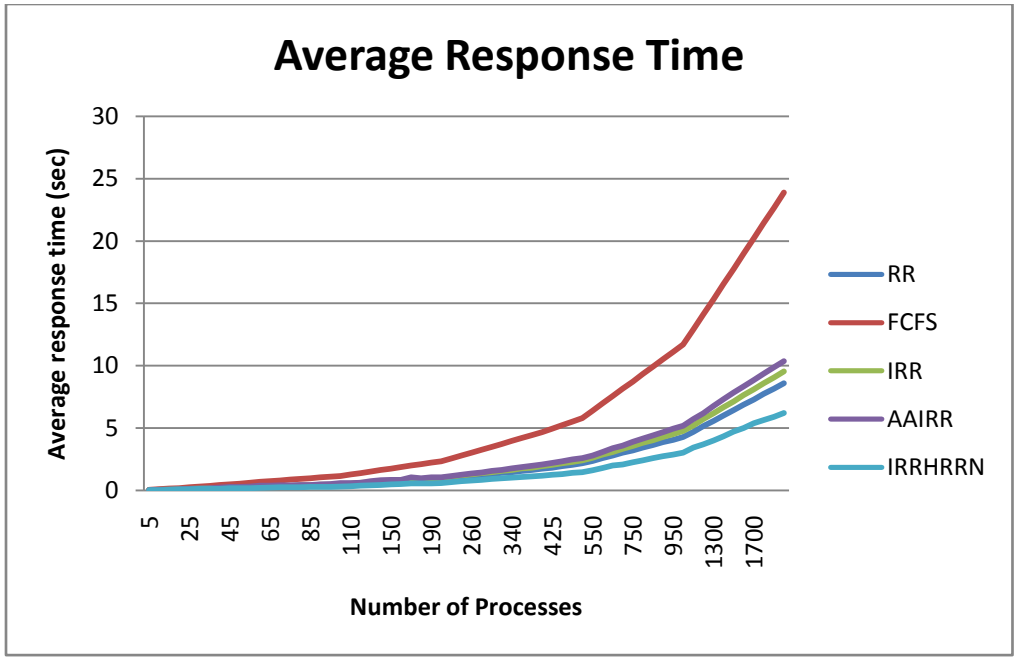
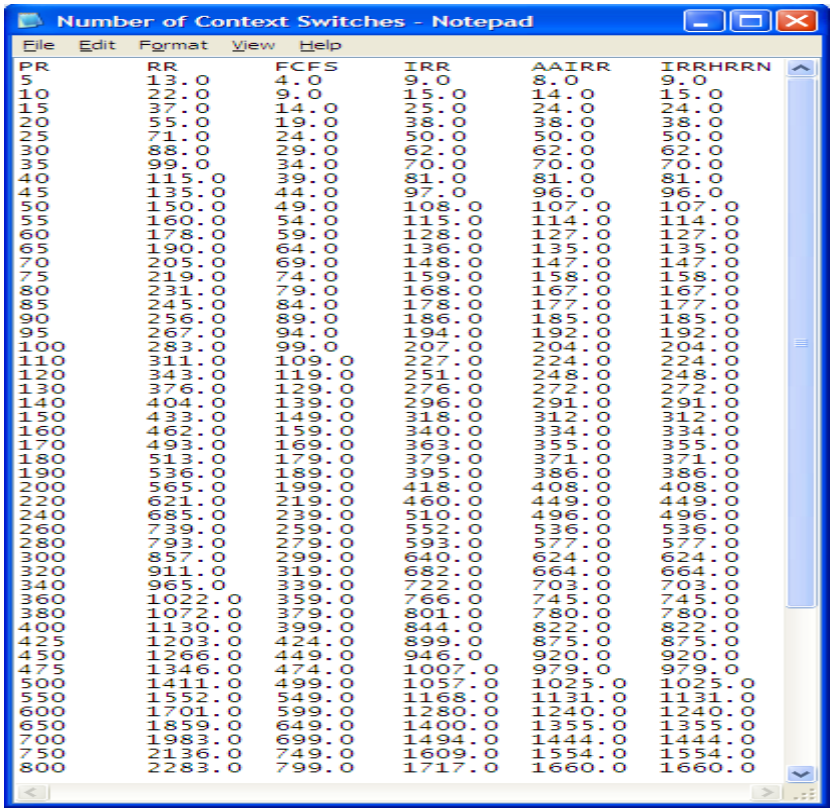


Figure 4.4: Graph of Average Response Time

Table4.5 presents some of the results of Number of Context Switches for the different times of runs and Figure 4.5 shows the overall graphical result of the Number of Context Switches for all cases of values taken. It was observed from the graph that the FCFS has the best performance, and it was followed by our IRRHRRN, AAIRR and IRR have almost the same performance at the second position, then RR followed.

Table 4.5: Some of the Results of Number of Context Switches



PR	RR	FCFS	IRR	AAIRR	IRRHRRN
5	13.0	4.0	9.0	8.0	9.0
10	22.0	9.0	15.0	14.0	15.0
15	37.0	14.0	25.0	24.0	24.0
20	55.0	19.0	38.0	38.0	38.0
25	71.0	24.0	50.0	50.0	50.0
30	88.0	29.0	62.0	62.0	62.0
35	99.0	34.0	70.0	70.0	70.0
40	115.0	39.0	81.0	81.0	81.0
45	135.0	44.0	97.0	96.0	96.0
50	150.0	49.0	108.0	107.0	107.0
55	160.0	54.0	115.0	114.0	114.0
60	178.0	59.0	128.0	127.0	127.0
65	190.0	64.0	136.0	135.0	135.0
70	205.0	69.0	148.0	147.0	147.0
75	219.0	74.0	159.0	158.0	158.0
80	231.0	79.0	168.0	167.0	167.0
85	245.0	84.0	178.0	177.0	177.0
90	256.0	89.0	186.0	185.0	185.0
95	267.0	94.0	194.0	192.0	192.0
100	283.0	99.0	207.0	204.0	204.0
110	311.0	109.0	227.0	224.0	224.0
120	343.0	119.0	251.0	248.0	248.0
130	376.0	129.0	276.0	272.0	272.0
140	404.0	139.0	296.0	291.0	291.0
150	433.0	149.0	318.0	312.0	312.0
160	462.0	159.0	340.0	334.0	334.0
170	493.0	169.0	363.0	355.0	355.0
180	513.0	179.0	379.0	371.0	371.0
190	536.0	189.0	395.0	386.0	386.0
200	565.0	199.0	418.0	408.0	408.0
220	621.0	219.0	460.0	449.0	449.0
240	685.0	239.0	510.0	496.0	496.0
260	739.0	259.0	552.0	536.0	536.0
280	793.0	279.0	593.0	577.0	577.0
300	857.0	299.0	640.0	624.0	624.0
320	911.0	319.0	682.0	664.0	664.0
340	965.0	339.0	722.0	703.0	703.0
360	1022.0	359.0	766.0	745.0	745.0
380	1072.0	379.0	801.0	780.0	780.0
400	1130.0	399.0	844.0	822.0	822.0
425	1203.0	424.0	899.0	875.0	875.0
450	1266.0	449.0	946.0	920.0	920.0
475	1346.0	474.0	1007.0	979.0	979.0
500	1411.0	499.0	1057.0	1025.0	1025.0
550	1552.0	549.0	1168.0	1131.0	1131.0
600	1701.0	599.0	1280.0	1240.0	1240.0
650	1859.0	649.0	1400.0	1355.0	1355.0
700	1983.0	699.0	1494.0	1444.0	1444.0
750	2136.0	749.0	1609.0	1554.0	1554.0
800	2283.0	799.0	1717.0	1660.0	1660.0

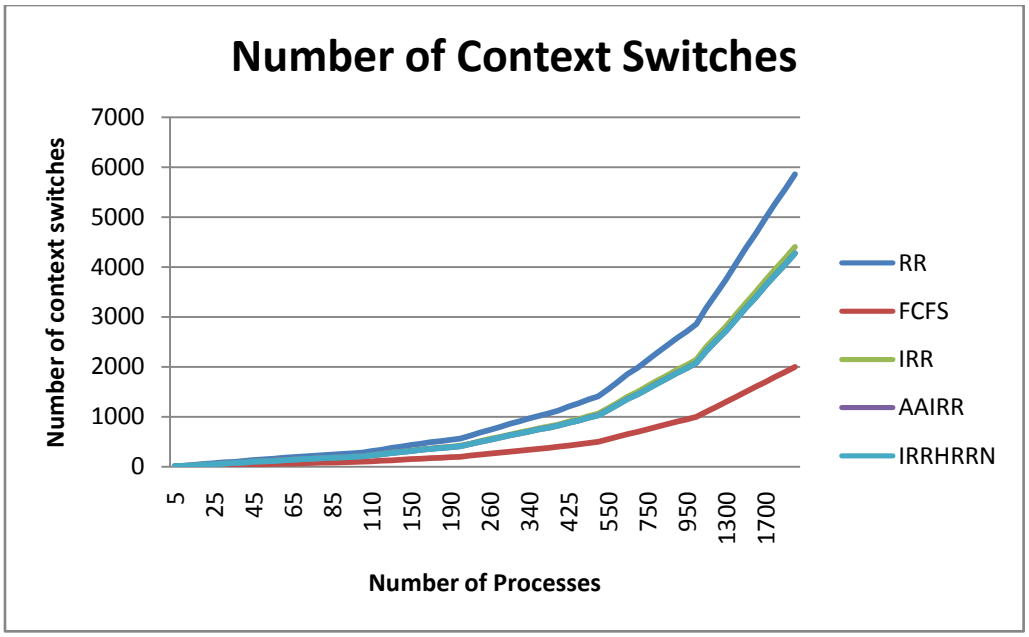


Figure 4.5: Graph of Number of Context Switches

From the above results, IRRHRRN produced the bestresults in terms of minimizing Average Waiting Time (AWT) and Average Turnaround Time (ATAT);AAIRR came second followed by FCFS, IRR andRR respectively. IRRHRRN has the best performance in terms of minimizing Average Response Time and this is followed by RR, IRR, AAIRR and FCFS respectively. Though it would seem FCFS has the best performance in terms of minimizing Number of Context Switches from figure 4.5, IRRHRRN, AAIRR and IRR actually has the best results, this is because in FCFS there is no context switches as the processes are released by the processor only when they have been completely executed. Therefore, IRRHRRN, AAIRR and IRR came first having almost the same performance with RR having the worse performance.

Table 4.6: Performance Percentage Comparing the Algorithms

Algorithms	ART	AWT	ATAT	NCS
RR	30.49%	38.71%	38.59%	65.39%
FCFS	74.93%	20.21%	20.10%	0%
IRR	37.43%	24.79%	24.69%	53.78%
AAIRR	43.37%	7.33%	7.29%	52.50%
IRRHRRN	0%	0%	0%	52.49%

Table 4.6 shows the performance percentage of the algorithms. 0% score shows that the algorithm is the best and 100% score shows that the algorithm is the worst in any of the categories compared. In terms of the Average Response Time (ART), IRRHRRN is the best while FCFS had the worst performance. RR which came second, performed better than IRR with 7.43%. AAIRR which came fourth was better than FCFS which was the last by 31.56%.

For the Average Waiting Time (AWT) category, IRRHRRN was the best and performed better than RR which recorded the worst performance with 38.71%. Closely trailing IRRHRRN was AAIRR with performance of the former better than that of the latter with just 7.33%. FCFS was better than IRR with 4.58%.

For Average Turnaround Time (ATAT), IRRHRRN still performed better than others with almost the same situation playing out as with the Average Waiting Time category. For Number of Context Switches (NCS), FCFS recorded the best results. This is because processes are not preempted, once theygain access to the CPU, they do not release it until they are completely executed. Both IRRHRRN and AAIRR performed well with the former doing better than the

latter with just 0.01%, and IRR closely followed AAIRR. RR had the worst performance in this category.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

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

5.1 Summary

Round Robin CPU scheduling algorithm is the most suitable scheduling algorithm used in timesharing operating systems, but its performance 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.Some improvements have been in the area of static provision of time quantum in Round Robin CPU Scheduling algorithm.This thesis presents an algorithm that made improvements on the Improved Round Robin scheduling algorithms by Manish and AbdulKadir(2012) and Abdulrahim*et al.,* (2014) by adopting the principle of Highest Response Ratio Next (HRRN) (Behera*et al.*, (2012)). Its goal is to increase the performance of the Round Robin CPU scheduling algorithm by reducing the average waiting time, number of context switches, average response time and average turnaround time. This algorithm (IRRHRRN) together with the simple RR,FCFS, IRR and AAIRR CPU scheduling algorithms were simulated using Java Programming Language and their results were compared based on AWT, ATAT, ART and NCS for different categories of processes that were generated randomly (i.e. uniform distribution for burst time and Poisson for arrival time).

5.2 Conclusion

The simulation results show that our algorithm (that is, IRRHRRN) was the best scheduling algorithm in terms of minimizing AWT and ATAT, and it was followed by AAIRR,FCFS, IRR andRR respectively. IRRHRRN has the best performance in term of minimizing Average Response Time and this was followed by RR, IRR, AAIRR and FCFS respectively. FCFS has the best performance in terms of minimizing Number of Context Switches (because when a process is allocated to the processor, it retains the processor until it finishes its execution), and the proposed IRRHRRN, AAIRR and IRR came second having almost the same performance, and were followed RR.

Based on the results obtained, it was observed that the performance of the this algorithm (IRRHRRN) was better than that of the Simple Round Robin, IRR and AAIRR CPU scheduling algorithms, in the sense that, it produces minimal average waiting time, average turnaround time, average response time and number of context switches. And it should be preferred over these algorithms.

5.3 Recommendation

This algorithm (that is,IRRHRRN) is recommended in the systems that adopt the Round Robin scheduling whose time quantum are provided statically; so as to improve the performance of the systems.Aside from the improvementsin the performance of Round Robin CPU scheduling algorithm, the main line of future research work is to dynamically determine the time quantum to be used in this improved algorithm.

REFERENCES

Abdulrahim A, Aliyu S, Mustapha A. M and Abdullahi S.E (2014): An Additional Improvement in Round Robin CPU Scheduling Algorithm, *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 4(2), pp. 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, 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.

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 (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-3, 200-206

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

Gerald, S., Garima, K., and P, S. (2004). Job Fairness in Non-preepmtive Job Scheduling. *International Journal on Parallel Processing* , 186-194.

Ishwari S. R. and Deepa G. (2012): A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems, 1-11.

Krishnan P, (1995). Online Prediction for Databases and Operating Systems.*Phd Thesis from Brown University, Providence Rhode Island* Retrieved November, 2013ftp://128.148.32.111/pub/techreports/95/cs95-24.pdf

Manish K. M. and Abdul Kadir K. (2012): AN IMPROVED ROUND ROBIN CPU SCHEDULING ALGORITHM,*Journal of Global Research in Computer Science,* Volume 3, No. 6,64-69

Mohammed Abdullah Hassan Al-Hagery (2011). A selective Quantum of Time for Round Robin Algorithm to Increase CPU Utilization. *International Journal of Computer Information Systems, Vol. 3, No. 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 Shenoy, (Fall,2008).*CPU Scheduling Lecture Slides*. Retrieved November 1, 2013, from UMASS Operating System

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

Silberschatz, P. B. Galvin, and G. Gagne (2006), "Operating System Concepts", 7th Edn., John Wiley and Sons Inc, ISBN 0-471-69466-5

Soraj H., and Roy K.C. (2011): Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice", *International Journal of computer Science and Communication*, 2(2), 219-326.

Stallings, W. (2001). *Operating Systems, Internals and design Principles* (4th ed.). Prentice Hall 2001, ISBN 0-13-031999-6 CA, USA

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) Volume 49– No.17, July 2012,* 4-9