OPTIMIZED ROUND ROBIN SCHEDULING

T07786

Research Dissertation submitted by:

Muhammad Sheeraz
Reg. No. 304-FAS/MSCS/F06

Supervised by:

Mr.Qaisar Javaid
Assistant Professor
Department of Computer Science
International Islamic University, Islamabad.

Department of Computer Science
Faculty of Basic and Applied Sciences
INTERNATIONAL ISLAMIC UNIVERSITY
ISLAMABAD
2011

Submitted to the department of Computer Science in partial fulfillment of the requirements
for the degree of
Master of Science
in
Computer Science

Department of Computer Science

INTERNATIONAL ISLAMIC UNIVERSITY ISLAMABAD

Dated: ______________

Final Approval

It is certified that we have examined the thesis project report titled "Optimized Round Robin Scheduling" submitted by Mr. Muhammad Sheeraz, Reg. No. 304-FAS/MSCS/F06 and it is our judgment that this research project is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad for awarding the degree of Masters of Science in Computer Science MS (CS).

Committee:

External Examiner:
Dr. Waseen Shahzad
Assistant Professor
Department of Computer Science
FAST, Islamabad.

[illegible]

Internal Examiners:
Mr. Asim Munir
Assistant Professor
Department of Computer Science
International Islamic University, Islamabad

[illegible]

Supervisor:
Mr. Qaisar Javaid
Assistant Professor
Department of Computer Science
International Islamic University, Islamabad.

[illegible]

Declaration

I hereby declare that this research dissertation is not copied as a part or whole from any source. It is further declared that no portion of this report has been submitted in support of any other degree or qualification of this to any other university or institute of learning.

Mr. Muhammad Sheeraz
304-FAS/MSCS/F06

Acknowledgements

The whole praise is to Almighty Allah, the most beneficent, the most merciful. Who gave me vision and enabled me to accomplish this dissertation.

To my supervisor Mr. Qaisar Javaid for giving me his precious time and guidance for completing this dissertation.

To my parents and family for their continuous and never ending invocations, cooperation and encouragement. Their prayers for my success are an endless source of encouragement for me in all spheres of life.

Dedication

I would like to dedicate it to my parents and family.

Abstract

Round robin scheduling algorithm is widely used for the purpose of scheduling. The reason for this is that round robin scheduling algorithm is very suitable for multitasking environment. In a multitasking environment more than one process is being executed concurrently. Round robin scheduling algorithm gives CPU time to each task in a circular fashion.

Round robin scheduling algorithm is widely used in field like operating system, network devices etc. In operating system round robin scheduling algorithm is used for scheduling of the processes. In network devices it is used to schedule network packets for processing.

But there is a potential problem associated with this classical scheduling algorithm. If a small process or task is present at the end of the queue, this task or process has to wait for quite a long time because all the processes present in front of it get CPU time before this process. In this situation the waiting and turn around times of this task or process are increased very much. Even if a large task or process is present at the end of the queue, again it has to wait for quite a long time, which increases its waiting and turn around times. If this problem is resolved the classical round robin scheduling algorithm can be made more optimized and useful.

This thesis is based on resolving this issue of round robin scheduling algorithm. A new scheduling algorithm named as "Optimized Round Robin Scheduling Algorithm" has been proposed in this thesis. This new proposed algorithm is based on the classical round robin algorithm with some variations. The above mentioned issue of round robin scheduling algorithm has been resolved in the newly designed algorithm by decreasing average waiting and turn around times of the processes. This algorithm specially looks after the process at the end of the queue and guarantees that its waiting and turn around times do not increase very much. For testing purpose the new proposed algorithm has been applied on sets of processes. This has shown that the new proposed algorithm is more efficient and optimized by exhibiting low average waiting and turn around times for the processes, as compared to the classical round robin scheduling algorithm and some of its variants discussed in this dissertation.

Optimized round robin scheduling algorithm is also applicable to network devices for scheduling network packets and in any other field in which scheduling is involved.

TABLE OF CONTENTS

Chapter No. **Contents** **Page No.**

1. Introduction 1
1.1. Process management 2
1.1.1. Process 2
1.1.2. Process States 3
1.1.3. Process Control Block 5
1.1.4. Process Scheduling 7
1.1.4.1. Scheduling Queues 7
1.1.4.2. Schedulers 8
1.1.4.3. Context Switch 9
1.2. CPU Scheduling 10
1.2.1. Scheduling Criteria 10
1.2.2. Scheduling Algorithms 12
1.2.2.1 First Come First Served Scheduling 12
1.2.2.2 Shortest Job First Scheduling 12
1.2.2.3 Priority Scheduling 13
1.2.2.4 Round Robin Scheduling 14
1.2.2.5 Multilevel Queue Scheduling 14
1.2.2.6 Multilevel Feedback-Queue Scheduling 15
1.3. Contribution of this Dissertation 16
1.4. Dissertation Organization 16

2. Literature Survey 18
2.1. Introduction 18
2.2. Previous Work 18
2.2.1. Deficit Round Robin 19
2.2.2. iSLIP Scheduling 19
2.2.3. Start-time Fair Queuing 20
2.2.4. Two Dimensional Round Robin Scheduler 20
2.2.5. Pre-order Deficit Round Robin 20
2.2.6. Stratified Round Robin 21
2.2.7. Improved Algorithm for Weighted Round Robin 21

3. Problem Statement And Proposed Solution 23
3.1. Introduction 23
3.2. Problem definition 23
3.3. Proposed Work 24
3.4. Optimized Round Robin Scheduling 25
3.4.1. Basic Theory 26
3.4.2. Explanation 26

3.4.3. Optimized Round Robin Algorithm 31
3.4.4. Case Scenarios 32
3.4.4.1. Best Case Scenario 32
3.4.4.2. Worst Case Scenario 32
3.4.4.3. Average Case Scenario 32

4. Testing and Performance Evaluation 33
4.1. Introduction 33
4.2. Performance 33
4.2.1. Process Set 1 34
4.2.2. Process Set 2 37
4.2.3. Process Set 3 38
4.2.4. Process Set 4 39
4.2.5. Process Set 5 40
4.2.6. Process Set 6 42
4.2.7. Process Set 7 43
4.2.8. Process Set 8 44
4.2.9. Process Set 9 46
4.2.10. Process Set 10 47
4.3. Performance Evaluation of Optimized Round Robin 49
4.3.1. Graph for Waiting Time 49
4.3.2. Graph for Turn around Time 50
4.3.3. Overall Performance 51

5. Conclusion and Future Work 55
5.1. Introduction 55
5.2. Conclusion 55
5.3. Future Work 56

References 57

LIST OF FIGURES

Figure 1.1:	Process Sates
Figure 3.1:	Execution of Optimized Round Robin algorithm
Figure 4.1:	Graph showing waiting times of 10 sets
Figure 4.2:	Graph showing turn around times of 10 sets
Figure 4.3:	Graph showing overall performance

CHAPTER 1

INTRODUCTION

Operating system is a computer programme like other computer programmes in a computer system. The difference between operating system and other computer programme is that the operating system looks after all other programmes and those programmes are executed under the umbrella of operating system. Other computer programmes cannot execute on their own which means that they are dependent on operating system. Therefore for the successful execution of other applications or programmes computer system must have an operating system. It is the first programme that is loaded into the machine.

An operating system is a platform through which users can launch and execute their programmes and take advantage of the resources of a computer system. In other words an operating system is a middle or intermediate programme that acts between the hardware resources of a computer and the users of the computer system or application programmes. Operating system in itself is basically a programme that manages all the resources of a computer system.

An operating system does some basic tasks such as

- Recognizing the input from input devices like keyboard or mouse
- Sending output to the output devices like monitor
- Keeping track of files and directories on the disks and
- Controlling peripheral such as disk drives and printers

Basically an operating system is an integrated programme of different programme modules, like process management module, memory management module, I/O management module etc. All these modules perform their predetermined tasks.
The scope of my research work is related to process management module of operating system and more precisely the process scheduling algorithms working under process management module of operating system.

1.1. Process Management

In process management operating system creates and deletes processes [user and system], schedules them and provides means for their synchronization, communication and deadlock handling.

Process management module is the part of operating system that manages and schedules different processes for execution. Because in today's multitasking environment many processes are executed simultaneously, there should be a mechanism that can ensure the execution of more than one process simultaneously as well as processes should be scheduled such that any chance of starvation or deadlock is avoided.

Deadlock is a situation in which a process 'A' after occupying resource 'R1' needs another resource 'R2' that another process 'B' occupies and process 'B' after occupying resource 'R2' needs another resource 'R1' that process 'A' occupies. Because of this situation neither of the process will be able to continue its execution.

1.1.1. Process

The terms process and programme are often used interchangeably. But there a fine difference between these two terms. A programme currently being in execution by the CPU is called a process. Actually there is no difference between a programme and a process, because they both are pieces of code.

The only difference is that whether that piece of code is being executed by the CPU at present or not. If this piece of code is not being executed by the CPU at present it is called a programme. And this very piece of code is known as process if this piece of code is being executed by the CPU presently.

Hence a programme is static in nature because CPU is not executing it presently. Whereas a process is dynamic in nature because CPU is executing it presently. As process is being executed by the CPU it requires some address domain in memory, CPU time, files, some variables etc. and these attributes are associated with dynamic entities.

A process is considered as a unit of work in most of the systems. There are many processes that are being run at a time in a system some of these processes are system processes and some are user processes. System code is executed by system processes whereas user code is executed by user processes.

Historically a process consists of only one thread of control but most modern operating system support processes that have more than one thread i.e. multithreaded process.

1.1.2. Process States

Process state is the stage of a process that is being executed by the CPU. The eligibility of the processes for receiving CPU time is determined by these states. These process states are not actually recognized by the operating system as we do but these states are very useful abstraction for comprehending processes.

These process states basically show the stage of the executing process. These process states enable us better understand the execution life cycle of a process under execution.

A process can be identified in an operating system by a state among the following;

Started (New)

Terminated

Ready

Waiting (Blocked)

Running

These states are shown graphically in the following diagram.

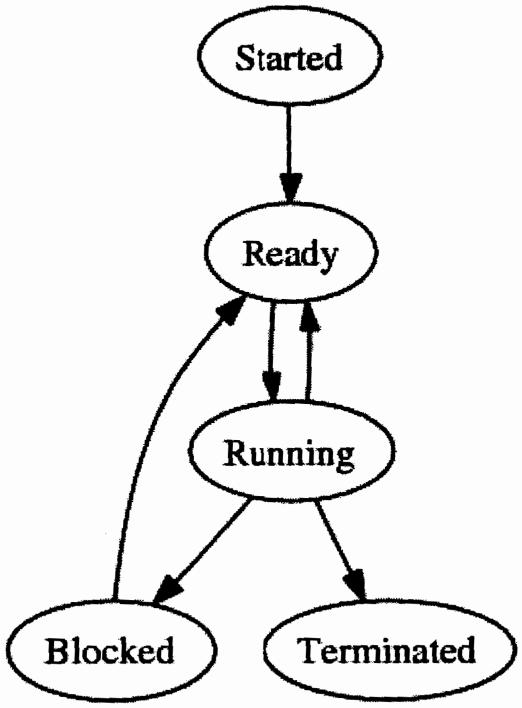

Figure 1.1 Process Sates

Started

Upon creation of a process it is in "Started", "Created" or "New" state. Process waits for admission to ready queue during this state. Long Term Scheduler either approves or delays the request to be admitted to ready queue by the process. This request of the process is automatically approved without any delay in normal desktop systems but this request can be delayed in real time systems. The reason for this delay is that the approval of this request may lead to over saturation. This will decrease the performance of real time systems which is not acceptable.

Ready

During this state the process waits for getting execution time from CPU after it gets loaded into main memory i.e. it is waiting for context switching by the Short Term Scheduler which is also known as Dispatcher. More than one process can be in ready state. Especially in case of single processor architecture many processes are present in the ready state and waiting to get CPU execution time. As there can only be one process in running state by the CPU.

Running

A process is known to be in running state which is presently in execution of CPU. There can only be one process in running state because more than one process can not be handled by the processor at one time.

Blocked

If a process waits for occurrence of an event e.g. completion of I/O operation or some signal, then it is in blocked state. Without getting its event completed this process cannot proceed further.

Terminated

A process can proceed to terminated state by two ways.

1. from running state after going to completion
2- be killed explicitly

It is very important to remove this process from memory after it completes its execution. If it resides in memory it will occupy its memory address domain. Such a process is known as Zombie process.

Additional Process States

The systems with virtual memory support have two additional states.

Swapped out and waiting

Mid Term Scheduler may swap out a process of the main memory and place it in virtual memory. Main memory can have this process swapped back in ready state.

Swapped out and block

Mid Term Scheduler can swap out blocked processes. Such processes may again be sapped back into main memory in the same way as swapped out and waiting processes.

1.1.3. Process Control Block (PCB)

The structure through which processes are represented in an operating system is called Process Control Block. It is called Task Control Block as well. It contains different attribute values related to processes. Some of the attributes are mentioned below;

Process Sate

It shows the state of the process.

Process Number

This number associated to each process by the operating system distinguishes a process from others in the system.

Programme Counter

PCB also contains an attribute programme counter. Programme counter contains address of an instruction. CPU executes this instruction next.

CPU Registers

Depending upon computer architecture there are different number and types of CPU registers. They include accumulator, index registers, stack pointers and general purpose registers plus any condition code information. It is vital to save the contents of these registers along with programme counter whenever an interrupt occurs for proper execution of the process.

CPU Scheduling Information

It contains a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory Management Information

Values for base and limit registers are contained in this information. Apart from that page or segment table value are also present here.

Accounting Information

It includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

I / O Status Information

This attribute contains information like set of I/O devices acquired by the process. It also contains a list of open files and so on.

1.1.4. Process Scheduling

The very purpose of multiprogramming is to use CPU in such a way to have its maximum advantage. Switching CPU among different processes very fast so that users can be very comfortable while interacting with running processes is the main purpose of time sharing. For this purpose process scheduler selects an available process for execution on CPU. There can be more than one process that is competing for CPU time.

This is the job of process scheduler to select a single and suitable process for execution by the CPU. In case of a single processor system there can be only one running process at a time. If multiple processes are present then one process will be running and rest will wait for their turn.

1.1.4.1. Scheduling Queues

The operating system contains some queues. These queues are used to hold different processes during their different stages in the system. Operating system contains a job queue. All processes of the system are contained in Job queue.

A process present in the main memory being ready to get its turn on CPU is kept in a queue known as ready queue. Linked list is the structure in the form of which this queue is normally implemented. First and last PCBs are pointed to by pointers contained in this queue header. A pointer is contained in every Process Control Block pointing to next PCB in queue.

A process enters into running state from ready queue. Now this process can complete its execution, can be interrupted or it can wait for some event like I/O completion. A device queue is attached to each I/O device in which processes that are waiting for some I/O are placed.

1.1.4.2. Schedulers

A process during its lifetime migrates among different system queues. Operating system is responsible for selecting appropriate process from these queues and hand it over to CPU. The process of selecting an appropriate process is performed by a module of operating system known as scheduler. The types of schedulers working in an operating system are discussed below.

Quite often in an operating system, the number of processes submitted is greater than the number of processes that can get CPU time. Such processes are placed traditionally on a disk for execution in future.

The process of selecting processes from here and loading them into main memory is carried out by long term scheduler.

The process of selecting one process from among these processes and allocating CPU to it is performed by short term scheduler.

The frequency of execution of these two schedulers is a primary distinction between them. Short term scheduler undergoes execution in greater number as compared to long term scheduler. A process must be selected for execution from ready queue by short term scheduler very frequently. Short term scheduler often executes once in 100 milliseconds. Therefore short term scheduler must be very fast in its selection process. Because if short term scheduler is not fast, as it executes very frequently, much of CPU time will be wasted in executing short term scheduler.

On the other hand the execution of long term scheduler is much less in frequency. There can be gap of minutes in creation of processes. The degree of multiprogramming i.e. process in memory is controlled by long term scheduler.

More time can be taken by long term scheduler in deciding which process to execute, because it executes in less frequency. Long term scheduler should very carefully select a process for execution.

Another intermediate scheduling is added in some operating systems like time sharing systems. Medium term scheduler performs this scheduling. At times it is beneficial to eliminate a process from the memory. The process that is eliminated from the memory can be brought back and it can restart its execution from that point it left. This technique is known as swapping. It is performed by medium term scheduler.

1.1.4.3. Context Switch

When CPU executes a process quite often in an operating system some sort of interrupt occurs. Because of this operating system is required to save the context of the process that is being executed presently. This is because of the fact that if the current state of the process that is stopped is saved then its execution can start at a later time. This technique of saving and at a later stage restoring the context (state) of a process by the CPU is known as context switch.

If current state of a process which is stopped because of an interrupt is not saved then in this case the execution of this process can not start from the point it was stopped. Therefore this is very important to save its current state.

PCB saves the state of a process which is executing currently when it is context switched and later when this process gets its turn again, its saved state is loaded again and execution starts from the point where this process get stopped. Hence a CPU state save is performed and then CPU state restore is performed in this process.

Therefore a computing process in which a process in which CPU state is stored and then some other task is performed. Later CPU starts execution from its saved state. The benefit of this technique is that a single CPU is managed in a way that it executes more than one process. Context Switch is an overhead and it should not occur too frequently because in this way most of CPU time would be lost in this process. Therefore although context switch is very usefull technique but it should be used very carefully.

1.2. CPU Scheduling

Scheduling is a basic technique today's operating system. Because we normally have more processes than available processors. Therefore it is important to schedule these processes so that they can be executed. This scheduling process is performed by a scheduler and dispatcher.

In case of a single processor system, there is only one processor is available to execute only a single process. But there are many processes that are present in such a system every time. These all processes require CPU time. This is the responsibility of operating system to ensure all processes get their turn and none of them is starved for an indefinite period. The basic concept of multiprogramming lies in the fact that CPU should be utilized efficiently by not allowing it to remain idle. Therefore CPU scheduling becomes very important so that all these processes are scheduled in a fashion such that they get their turn plus CPU is also utilized efficiently. This makes sense that whenever a system has many processes, they are needed to be scheduled by operating system for there complete execution.

1.2.1. Scheduling Criteria

Many CPU scheduling algorithms are available for the purpose of CPU scheduling. Each algorithm has its own traits and it might support a particular class of processes more than another class of processes. While selecting a particular scheduling algorithm traits of different scheduling algorithms should be considered. For the purpose of comparison between scheduling algorithms different criteria are available.

One algorithm might show best result for one criterion and it might show poor results for another criterion. Hence selection of an algorithm for a particular criterion is very important. Some criteria are discussed below;

CPU utilization

CPU is desired to be kept busy to the maximum. Theoretically CPU utilization should be between 0 to 100 percent. But in reality, it should range from 40 (in case of lightly loaded system) to 90 percent (in case of heavily used system).

Throughput

Work is being done when CPU is executing processes. Throughput is the number of processes executed per unit time. For long processes, this rate may be one process per hour; for short transaction, it may be 10 processes per second.

Waiting Time

The CPU scheduling algorithm cannot do any thing with time of execution of process or amount of time of doing I/O, it can only affect the amount of time that a process spends in ready queue called waiting time. Scheduling algorithm should try to minimize waiting time of a process.

Waiting time = time spent in ready queue by the process

Turnaround Time

Turnaround time is the time from submission of a process to its completion. Turnaround time is the sum of the time spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

Turn around Time = Burst time + Waiting time

Response Time

It is more useful criterion than turnaround time in an interactive system. Time form the submission of a request until the first response is produced is called response time.

CPU utilization and throughput criteria are tried to be maximized while turnaround, waiting and response times are tried to be minimized.

1.2.2. Scheduling Algorithms

Many scheduling algorithms are available and each of these algorithms has its own properties. They make it possible for us to execute many processes and prohibit any resource starvation in the system. Some of the very common CPU scheduling algorithms are described below briefly.

1.2.2.1. First Come First Served Scheduling

"First Come First Served" scheduling algorithm is the simplest scheduling algorithm. Here CPU is given to process that requests it first. And this process holds the CPU till its completion. This process does not give CPU to any other process until it completes it execution.

Scheduling overhead is minimal in this scheduling scheme because context switching is only performed when a process completes its execution not in between. Large process might hold CPU for long time therefore throughput of this scheduling algorithm might be low. Turnaround time might be high because of the very reason mentioned earlier. Waiting and response times might be high because of same reason. This scheduling algorithm permits every process for completing its execution and hence there is no starvation in this scheduling algorithm. There isn't any concept of prioritization. Hence this scheme faces problem meeting process deadlines. [22]

1.2.2.2. Shortest Job First Scheduling

This scheme takes different approach towards solving the processes. This scheduling algorithm focuses on the lengths i.e. amount of CPU time required by the processes. Shortest job first scheduling sorts all ready queue processes and picks up the process that requires the least CPU time, this process is solved to its completion and then the next process with least CPU time requirement is picked up and executed.

Theoretically this scheduling algorithm gives the most optimum results possible. At short term scheduler level this scheduling algorithm is not implement-able. As it is impossible to find exact CPU time that a process requires prior to its execution. One approach is to approximate the CPU time required by the process but of course it would be a predicted value and no guarantee can be given about the predicted value i.e. whether this value be accurate or not.

Shortest job first scheduling algorithm gives optimum average waiting times for the processes. Because it executes short process prior to long process therefore waiting times for the processes are decreased quite a bit. If a long process gets executed after a short process, waiting time of short process is decreased more than waiting time of long process is increased.

It is worth noticing that a more appropriate term for this scheduling algorithm is shortest next CPU burst scheduling algorithm. This algorithm can support preemption or it can be without preemption. Shortest remaining time first is another name that is given to the preemptive shortest job first scheduling algorithm. [22]

1.2.2.3. Priority Scheduling

This algorithm assigns priority value to every process. Process having the greatest priority is given CPU. This means that the processes with higher priorities are executed first always.

Interactive processes can be assigned higher priority values so that they can get a high response time. This algorithm is very useful because in some scenarios it is desirable to give much of CPU time to a specific process. Through priority scheduling algorithm this task is performed very efficiently.

This algorithm has two flavours preemptive and non-preemptive. In preemptive version of this algorithm currently executing process will be preempted if a process with greater priority arrives in queue and latest arrived process with greater priority will be assigned CPU.

Shortest job first scheduling algorithm is form of this algorithm where priority is amount of CPU burst time. If CPU burst time is higher, priority of this process will be lower.

There is chance of indefinite starvation for low priority processes in this algorithm. This scenario is created if high priority processes are continuously coming in the system. In this case a process in the system with a low priority will be blocked indefinitely. This process will be blocked till new processes with higher priority are coming. When processes with higher priority stop coming in the ready queue, only then this blocked process will have a chance to get CPU and start its execution.

A solution to this problem i.e. blocking for low priority processes is aging. In aging priority of process present in queue increases gradually with the passage of time. In this procedure a time comes when this process is assigned the highest priority in the system. Hence this process will have its turn on the CPU. [22]

1.2.2.4. Round Robin Scheduling

In time sharing systems processes should get equal amount of CPU time. This algorithm is specifically developed for time sharing systems. In this scheduling algorithm CPU time is given to all processes in a circular manner. This scheduling algorithm is basically working on the principle of first come first served scheduling except preemption is an added phenomenon.

This algorithm has a concept of time quantum that is the maximum time for which a process can execute on the CPU.

Queue is viewed like circular queue. All the processes one by one in a circular fashion are allocated the CPU. The processes are allocated CPU for a maximum time of one time quantum. After that either the process goes to its completion or it is put back in ready queue and CPU it allocated to the next process.

Time quantum value normally lies between ten to hundred milliseconds. The value of time quantum should not be very large. In this case round robin scheduling algorithm will be degenerated to first come first served scheduling. Similarly time quantum value should not be very low because in this case context switching will occur very frequently and much of CPU time will be wasted in context switching. Hence time quantum value ought to be greater than context switch time. As a rule the time quantum value should be chosen such that 80 percent of processes in queue get completed when they are executed first time for one time quantum. [22]

1.2.2.5. Multilevel Queue Scheduling

This algorithm is used in scenarios where processes can be divided into different groups. This division can be based on memory requirement, CPU time requirement, I/O requirement etc. For example consider foreground and background processes. These two group of processes have different resource requirments.

Multilevel queue scheduling algorithm comes up with the solution. It divides the ready queue into several queues. Each of these queues can follow a different scheduling algorithm. Assignment of processes to queues is permanent which is based on a specific property of process.

For example we are using two queues for foreground processes and background processes. Former require quick response therefore round robin scheduling algorithm would be a better choice to be used in this queue. On the other hand for background or batch processes first come first served scheme can be used.

Another variation that can be done is that the CPU time may be distributed among different queues according to the requirements of the processes. For example for foreground or interactive processes we can allocate say 70 percent of CPU time where for background or batch process 30 percent of the CPU time can be allocated. [22]

1.2.2.6 Multilevel Feedback Queue Scheduling

The scheme discussed above is bit inflexible as processes can not move between queues i.e. they do not change their queue during their execution. This scheme has an advantage that its overhead is low and foreground and background processes do not change their nature during their execution. But this scheme is inflexible.

Multilevel feedback queue scheduling algorithm is flexible in the sense that processes can move between different queues during their execution. The idea behind this scheme is to group processes with similar requirement. In this scheme if a process is using too much CPU time it is placed in low priority queue.

I/O bound and interactive processes are placed in higher priority queues. A process may be moved to higher queue that has been waiting for quite a long time in a low priority queue. It will avoid indefinite blocking or starvation for the process. [22]

1.3. Contribution of this Dissertation

This dissertation is focused on scheduling algorithms. There is a potential problem associated with this classical round robin scheduling algorithm. If a small process or task is present at the end of the queue, this task or process has to wait for quite a long time because all the processes present in front of it get CPU time before this process. In this situation the waiting time of this task or process is increased very much. Even if a large task or process is present at the end of the queue, again it has to wait for quite a long time, which increases its waiting time. This issue of round robin scheduling algorithm has been resolved by developing a new algorithm i.e. optimized round robin scheduling algorithm in this dissertation. Optimized round robin scheduling algorithm solves this issue by decreasing average waiting and turn around times of the processes.

Round robin algorithm is widely used in scheduling problems in today's multitasking environment. In multitasking environment many interactive processes are competing to get the CPU time. Therefore by resolving this specific issue associated with round robin scheduling algorithm, would prove very beneficial for scheduling environment where round robin algorithm is widely used.

Beauty of the dissertation is that the proposed algorithm does not have large computational complexity which is very essential in developing algorithms. Another key factor is that this algorithm is not specific to one field. Although in this dissertation the proposed algorithm is applied to processes of operating system but this is not the only place where this algorithm can be applied. This algorithm is also applicable in scheduling network packets in network devices etc.

1.4. Organization of Dissertation

The dissertation is composed of five chapters. These chapters elaborate key concepts and different aspects of scheduling. The first chapter named "Introduction" discusses introductory concepts that have been used in this dissertation.

Second chapter named "Literature Survey" discusses different literature regarding to this dissertation and different algorithms that have been developed in the past to resolve different scheduling issues. This literature survey focuses on different algorithms that are also implemented in other fields such as in network field.

Third chapter named "Problem Statement And Proposed Solution" discusses the issue associated with the round robin scheduling algorithm. This dissertation is basically based on this issue. Then this chapter discusses proposed solution, its overall concepts and theory. This chapter discusses in depth the new proposed algorithm, its basic theory, working and different case scenarios.

Fourth chapter named "Testing and Performance Evaluation" focuses on performance evaluation of new proposed algorithm. In this chapter different sets of processes are executed with the different algorithms including new proposed algorithm and comparison is performed between the results of these algorithms. These experimental results reveal that the new designed algorithm exhibits better performance than classical round robin and other algorithms considered in this dissertation by exhibiting low average waiting and turn around times for processes.

Fifth chapter named "Conclusion and Future Work" presents overall conclusion and the future work in this direction.

CHAPTER 2

LITERATURE SURVEY

2.1. Introduction

Round robin scheduling algorithm is widely used in its classical form and there are many variations of this algorithm. This chapter discusses some algorithms that used round robin scheduling algorithm in some form.

2.2. Previous Work

As discussed earlier, many scheduling algorithms are available for the purpose of scheduling processes in an operating system so that these all processes get a chance to be executed on the CPU and share computer resources. These scheduling algorithms use different scheduling techniques to schedule many processes. So that each and every process is executed to its completion and the process gets a feeling as if CPU were under its control. This is the beauty of multitasking environment that many different interactive processes are executed having CPU shared among them.

Another worth mentioning point is that scheduling algorithms are not confined to operating system design only. Rather these scheduling algorithms are used in various different fields. In fact wherever a queue is being formed and a resource is being shared among different entities, there is a need of some scheduling discipline. These scheduling algorithms are also used in network devices. In network devices such as routers, switches, bridges etc. scheduling algorithms are used to schedule data packets. Every data packet needs some CPU time to be processed. As there are many data packets traveling through a network device at a given time, therefore these data packets also need to be scheduled through some scheduling algorithm.

Some of the most popular process scheduling algorithms includes first come first served scheduling (FCFS), shortest job first scheduling (SJF), round robin scheduling (RR), multilevel queue scheduling, multilevel feedback-queue scheduling etc.

In today's multitasking (execution of more than one process simultaneously) environment round robin scheduling algorithms is very often used. Round robin scheduling algorithm is able to execute more than one process by allocating them time slices or quantum. Then it cycles CPU through each process in queue.

2.2.1. Deficit Round Robin

Fair Queuing technique is the concept that has been emphasized in this research paper. Mr. M. Shreedhar and Mr. George Varghese mention in [15] that when there is more than one flow passing through a network device then fair queuing is the technique that enables the network resources to be fairly shared among all the flows.

The previous available schemes achieving nearly perfect fairness were expensive from implementation point of view. Also the per-packet required work by those techniques has been O (log (n)), n represents active flows. Similarly the other available schemes with cheaper approximation of fair queuing showed unfair behavior.

They propose a scheme, "Deficit Round Robin (DRR)" that obtains nearly perfect fairness with per-packet required work of O (1). This scheme can be applied in different scheduling problems. This DRR scheme is especially very attractive to be used in gateways and routers. DRR scheme is especially suited for Datagram networks i.e. networks operating on UDP protocol although DRR is a generic scheme and can be applied to all the networks. DRR is also applicable to scheduling environments where the whole job should be processed at a time not a part of the job. Similarly DRR is also applicable to load balancing problem.

2.2.2. iSLIP Scheduling

Mr. Nick Mckeown mentions in [16] that FIFO input queues are generally avoided because if these queues are used they reduce the achievable bandwidth. Some algorithms guarantee 100% throughput.

The proposed "iSLIP Scheduling" algorithm, iterative round robin, for uniform traffic it provides cent percent throughput property and it's also very easy for implementation in hardware.

The simulation results of these versions under light and pressure conditions are analyzed. Under non-uniform traffic conditions iSLIP algorithm does have the ability to avoid the starvation of any input queue by adapting to fair scheduling policy. Two versions for iSLIP algorithm are proposed i.e. iterative and non-iterative.

2.2.3. Start-time Fair Queuing

Mr. Pawan Goyal, Mr. Harrick M. Vin and Mr. Haichen Cheng state in [18] that "Start-Time Fair Queuing (SFQ)" provides efficient results in spite of change in the capacity of the server. From computational view point it is very efficient and it also exhibits smallest fairness measure. They claim that for integrated services networks start-time fair queuing is more suitable than weighted fair queuing and claim that SFQ is certainly more efficient in performance than self clocked fair queuing.

2.2.4. Two-Dimensional Round-Robin Scheduler

Mr. Richard O. LaMaire and Mr. Dimitrios N. Serpanos mention in [19] state that in a packet switch operating on the concept of multiple input queues the proposed algorithm achieves higher throughput in addition to fair access. In this algorithm for each output there is a queue maintained by each input port. Proposed algorithm Fairness properties are shown and analyzed. These properties are then compared with input and output queuing configurations. This shows that "2-D Round Robin Scheduler" exhibits the same throughput as that of output queuing. Basically two algorithms are suggested i.e. basic and enhanced 2-dimensional round robin.

Both of these algorithms provide the same saturation throughput as provided by the queuing architecture. Both of these algorithms can be implemented through logic components which makes them high speed providers in switch implementation.

2.2.5. Pre-order Deficit Round Robin

Mr. Shih-Chiang Tsao and Mr. Ting-Dar Lin mention in [21] that deficit round robin algorithm has large latency plus unfair behavior problems. They adopt a technique that eliminates these problems by placing priority queues. This technique is same as of deficit round robin except certain number of queues are placed earlier to deficit round robin design and this reorders sequence of transmission.

They suggest "Pre-Order Deficit Round Robin (PDRR)" algorithm as a better alternative to deficit round robin by showing analysis on latency and fairness. They also show that it exhibits a complexity of O(1) and O(logZ) per-packet. They propose PDRR algorithm that tackles the problem of bursty transmission. This algorithm also solves the problem of inappropriate transmission sequence.

This algorithm does this by reordering the transmission sequence of packets. They state, in future, the behavior of PDRR is to be tested for different simulation cases so that its general behavior is observed better.

2.2.6. Stratified Round Robin

Mr. Sriram Ramabhadran and Mr. Joseph Pasquale state in [24] that fair queuing is a topic that has been given a fair amount of attention. But even then there remains a problem that some algorithms exhibiting better performance are very expensive to implement or are not feasible, and there are some algorithms that are not expensive to implement but their performance is not as desired. The "Stratified Round Robin" algorithm that is presented by them provides a low complexity solution. It does not matter how many flows are there this algorithm contains a unique property of single packet delay bound.

In stratified round robin algorithm flows that have almost same bandwidth requirements are grouped together into a single class of flow. With deficit weighted round robin technique is applied within a flow class. This scheme is very practical and feasible. Its implementation can be realized as a priority encoder. It is very feasible to use this technique in high speed routers because of its practicality.

2.2.7. An Improved Scheduling Algorithm for Weighted Round-Robin

Mr. Yao-Tzung Wung, Mr. Tzung-Puo Lin and Mr. Kuo-Chung Gan state in [30] that in designing an isochronous scheduler, priority and constant delay are considered as vital performance parameters. The solution proposed by them further enhances the performance of weighted round robin multiplexing which provides a very effective and simple technique for priority traffic. In an ATM switch, the smoothness performance in a weighted round robin multiplexer further improves by adopting this proposed technique.

This improved solution especially tackles the problems caused by delay jitters. Mean response time and its standard deviations are considered. This proposed solution by them is very much suited for the networks that are operating on real time applications. It will help improve quality of service of such networks.

The above discussed literature is based on the classical round robin scheduling algorithm in majority of the cases. This is because round robin scheduling algorithm is very widely used in operating system, network devices etc, Therefore if performance of round robin algorithm can be improved and optimized this would be very beneficial in many fields.

CHAPTER 3

PROBLEM STATEMENT AND PROPOSED SOLUTION

3.1. Introduction

This chapter discusses the typical problem associated with the classical round robin scheduling algorithm. This problem is defined and then proposed work has been discussed below.

In this chapter a new algorithm has been discussed which gives the solution to the problem associated with round robin scheduling algorithm. The proposed solution to the problem is discussed in detail with its basic theory and explanation.

3.2. Problem Definition

In today's multitasking environment interactive process requires constant CPU time. But because there are many processes running at a time in a multitasking environment, it is no feasible to hand over CPU to a single interactive process. The CPU time must be distributed among different processes present in the system. In this scenario round robin scheduling algorithm is very suitable to be used because round robin scheduling algorithm gives CPU one by one in a cyclic manner to all processes of queue. And every process in the queue gets CPU time.

Although round robin scheduling algorithm is very widely used in different fields now a days but a potential issue is there. If the ready queue used in round robin algorithm is too long, the process at end of queue waits for its turn for quite a long time. Although it might require a small portion of CPU time to complete its execution or even if it requires large portion of CPU time, it will execute for the specific time quantum and then it gets placed at end of queue again. It waits again for quite some time to get its turn. This will increase the waiting time for this process significantly.

This situation shows that there is a potential problem for the process at end of ready queue generally and specifically if a small process (requiring a small amount of CPU time for its execution) is present at the end of execution queue. These two situations are described below;

- If a process requiring small amount of CPU time is present at end of queue of round robin algorithm, say it requires 5 milliseconds of CPU time and say the time quantum is 50 milliseconds. But the problem is that this small process will have to wait for all the other processes in front of it to be executed either to their completion or equal to the time quantum. Although this process only requires a very small amount of CPU time but its waiting time is increased very much.

- The second situation arises if the process present at the end of ready queue of round robin scheduling algorithm requires large amount of CPU time. Say it requires 55 milliseconds and the time quantum is 50 milliseconds. Now this process will get its turn after all of its earlier processes either execute to their completion or they execute for the specific time quantum. And when it starts its execution it will execute for 50 milliseconds and then again it gets placed at end of queue. This process again waits and gets its turn after all the processes present before it are executed. Again in this case the waiting time for this particular process has increased to a great deal.

The above two situations clearly describe the specific problem that is associated with round robin scheduling algorithm.

3.3. Proposed Work

One solution that can be adopted is not to use round robin algorithm. But actually this cannot be a solution because for interactive processes round robin scheduling algorithm comes to be very suitable choice. So we cannot avoid using round robin scheduling algorithm for processes.

One another aspect is that round robin scheduling algorithm is not only used in operating systems for process scheduling, but round robin scheduling algorithm is also being used in network devices in processing network data packets. Hence round robin scheduling algorithm because of its wide and effective usage cannot be thrown away for solving the specific problem addressed above.

Therefore if a solution is provided for the above mentioned problem it would be very beneficial for scheduling purposes. This would not only be beneficial for process scheduling in operating system but it would also be beneficial for network devices that are using round robin scheduling algorithm for processing network data packets.

My proposed solution to the above mentioned problem is based on the features of classical round robin scheduling algorithm plus it more specifically considers the above mentioned problem. The solution focuses on the process that is present at end of queue. So that waiting time for the process that is present at end of queue does not increase very much rather its waiting time is be kept as minimum as possible. New proposed algorithm decreases average waiting and turn around times for the processes.

Once this issue relating to round robin algorithm gets resolved, round robin algorithm will become more efficient and optimized. The waiting times for the processes gets decreased and average waiting time for the processes in a specific time interval becomes less than the average waiting time for the same set of processes executed through round robin scheduling algorithm.

This proposed algorithm has been applied to processes. Actually this algorithm can also be applied to network devices i.e. in scheduling network packets. But because processes within an operating system are very common and easily understood therefore this new algorithm, round robin, deficit round robin and start time fair queuing scheduling algorithms have been applied to processes. After executing different sets of processes with these scheduling algorithms, the results of these scheduling algorithms have been compared.

The results have shown that this new proposed algorithm is more optimized and efficient by exhibiting low waiting and turn around times as compared to round robin, deficit round robin and start time fair queuing algorithms. Therefore the usability and effectiveness of round robin algorithm has been increased by this newly designed algorithm. This newly designed algorithm has been named as "Optimized Round Robin Scheduling Algorithm".

3.4. Optimized Round Robin Scheduling

Optimized round robin algorithm removes the specific problem associated with the processes discussed earlier.

Optimized round robin scheduling algorithm reduces the waiting time of the process present at the end of ready queue. It optimizes throughput, turnaround and waiting times. Actually in a multitasking and interactive environment waiting time is very important criterion and this waiting time is tried to be kept as low as possible so that every process in the system can get a feeling that CPU is dedicated to it entirely.

3.4.1. Basic Theory

Optimized round robin is an effective algorithm in time sharing systems where system must ensure low waiting time to interactive processes. The preemption overhead is kept low by effective context switching mechanisms and by providing adequate memory for the processes to remain in main memory simultaneously.

The basic theory of optimized round robin scheduling includes the following aspects;

- The processes are sorted according to their burst times.
- The process having smallest burst time is at the top of the ready queue.
- At the end of ready queue, process having largest burst time is placed.
- Two pointers are maintained in optimized round robin scheduling algorithm.
- The first pointer "tempLastBurstTime" keeps the burst time of the process present at the end of the ready queue. The process has the largest burst time.
- The second pointer "tempTotalBurstTime" stores the sum of burst times of executed processes.
- Every process gets a fixed time quantum in which either the process completes its execution or it executes for the time quantum and after that it is put at end of queue.
- After sorting first process is picked up and solved for a maximum of one time quantum.
- If time quantum is greater than process burst time it gets solved completely else it is placed at end of ready queue.
- The condition whether the "tempTotalBurstTime" is equal to or greater than the "tempLastBurstTime" (burst time of process at end of queue) is checked after executing a process for a maximum of one time quantum.
- If the condition is true then process present at the end of ready queue catches the CPU and is executed for a maximum of one time quantum. After that queue is again sorted and this fashion goes on.

3.4.2. Explanation

The optimized round robin scheduling algorithm works on sorting the processes according to their burst times i.e. the priority is assigned to process with the smallest burst time. Besides this, priority is also given to the processes with the large burst times. To remove the starvation of the processes with large burst times, two pointers are maintained i.e. "tempLastBurstTime" and "tempTotalBurstTime".

The pointer "tempLastBurstTime" stores the burst time of the largest process i.e. process present at end of queue. We can also say that this pointer points to the burst time of last process.

The pointer "tempTotalBurstTime" stores the sum of burst times of executed processes.

As optimized round robin scheduling algorithm is an advanced form of round robin algorithm, therefore it contains a time quantum as well. A process has to leave the CPU if time quantum has expired and the process has not executed completely. When a new process arrives that has shorter burst time, process that is executing currently doesn't leave the CPU but continues the execution until it completes its execution or time quantum expires.

The risk of starvation for the processes with larger burst time is removed by using the pointers mentioned above. The pointer "tempLastBurstTime" is compared against pointer "tempTotalBurstTime".

If pointer "tempTotalBurstTime" is greater than or equal to pointer "tempLastBurstTime" then process present at end of queue, having smaller or equal burst time to the value of the pointer "tempTotalBurstTime", is executed and the pointer "tempTotalBurstTime" is reset to 0. If the process is not executed to its completion it is again placed at end of queue.

Selecting time quantum value is same as method of selection of round robin algorithm. As a generic rule this value is chosen such that 80 percent of the processes are executed to their completion when they execute for one time quantum.

The ready queue is sorted whenever the value of "tempTotalBurstTime" becomes equal to or greater than "tempLastBurstTime".

The following figure explains this scheduling scheme in pictorial form. Here first of all process 'A' is picked up by the CPU and it executes for one time quantum after that it is put back at the end of the ready queue. The process 'A' is placed at the end of the queue because its burst time is greater than the time quantum. Process 'B' and process 'C' will gain CPU after that and after them process 'A' can have its turn on CPU. This fashion continues.

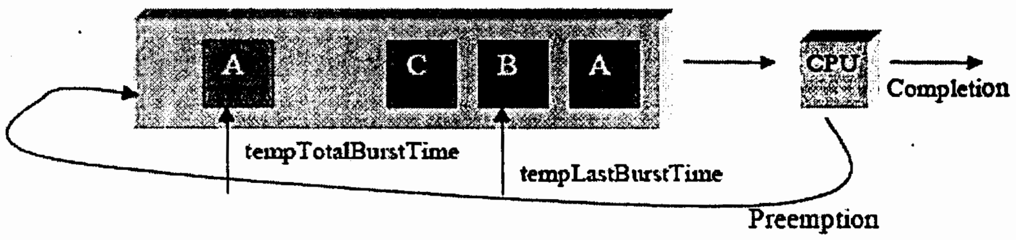

Figure 3.1 Execution of Optimized Round Robin algorithm

The working of optimized round robin scheduling algorithm is further explained with the help of an example. Let's take a process set. Their arrival and burst times are as follows. Time quantum for this example is taken as 10ms.

Process	Arrival Time	Burst Time
P0	0	8
P1	0	4
P2	0	6
P3	0	13
P4	0	1

It is assumed that all processes have arrived in queue at time 0. First of all queue is sorted and the order of sorted processes is;

P4 --- P1 --- P2 --- P0 --- P3

As P4 is first process in queue, execution is started from it. "tempTotalBurstTime" pointer gets initialized with value of 0 and "tempLastBurstTime" pointer is initialized with the value 13 i.e. the burst time of the last process in the queue. As P4 requires only 01 millisecond of CPU time therefore it executes to its completion i.e. its burst time is less than time quantum. After executing P4 "temptTotalBurstTime" pointer is updated with the value 1. "tempLastBurstTime" pointer holds the value 13.

After the execution of P4, P1 is given the CPU time as it is now the smallest process in the queue. It requires 4 milliseconds of CPU time which is less than time quantum therefore P1 also completes its execution. At this time "tempTotalBurstTime" pointer value is updated with the value 5 and "tempLastBurstTime" still contains the value 13.

After the execution of P1, P2 is given the CPU time as it is now the smallest process in the queue. It requires 6 milliseconds of CPU time which is less than time quantum therefore P2 also completes its execution. At this point "tempTotalBurstTime" pointer is updated with the value 11 and "tempLastBurstTime" still holds the value 13. After executing each process the values of both the pointers are compared to see whether the value of "tempTotalBurstTime" pointer is greater than or equal to the value of "tempLastBurstTime" pointer. But up to this point this condition has not become true. At this very point this condition is as follows;

tempTotalBurstTime >= tempLastBurstTime

11 >= 13

The above condition is not true yet therefore execution continues in normal fashion.

After the execution of P2, P0 is given the CPU time as it is now the smallest process in the queue. It requires 8 milliseconds of CPU time which is less than time quantum therefore P0 also completes its execution. At this point "tempTotalBurstTime" pointer is updated with the value 19 and "tempLastBurstTime" still holds the value 13. Now it is obvious that the above mentioned condition becomes true because the value of "tempTotalBurstTime" pointer is now greater than the value of "tempLastBurstTime" pointer. Therefore now CPU time will be given to process present at end of queue. At this point when the above mentioned condition becomes true the values of both pointers get updated. Pointer "tempTotalBurstTime" gets updated with the value 0 and the pointer "tempLastBurstTime" is updated with value of burst time of last process in queue, which happens to be 13.

Now P3 starts execution. It executes for 10 milliseconds and then preempted as time quantum expires and this process requires more CPU time. Because queue has only one process therefore it regains CPU time and completes its execution.

The Gantt chart of the execution of these processes is shown below;

P4	**P1**	**P2**	**P0**	**P3**	**P3**

0 1 5 11 19 29

32

The waiting times of the processes are given below;

P0 = 11 ms P1 = 1 ms P2 = 5 ms P3 = 19 ms P4 = 0 ms

Average Waiting Time = (11+1+5+19+0) / 5 => 7.2 ms

The turn around times of the processes are given below;

P0 = 19 ms P1 = 5 ms P2 = 11 ms P3 = 32 ms P4 = 1 ms

Average Turn around Time = (19+5+11+32+1) / 5 => 13.6 ms

Optimized round robin scheduling algorithm has shown better results compared to other algorithms discussed in this dissertation but optimized round robin scheduling algorithm does this at the cost of sorting. In optimized round robin sorting could occur frequently which consumes CPU time and this is overhead of optimized round robin scheduling algorithm that we have to bear.

3.4.3. Optimized Round Robin Algorithm

```
OptimizedRoundRobin ( processes, time_quantum )
    1.  Sort the queue in ascending order
    2.  n = processes
    3.  Loop: from 1 to n
            a. Solve the process up to one time quantum
            b. Check if ( tempTotalBurstTime >= tempLastBurstTime )
                    1. Solve process with the largest burst time up to one time quantum
                    2. Sort the queue
                    3. Go to step 3
            c. Sort the queue if new processes have arrived
            d. Go to step 3
End OptimizedRoundRobin
```

3.4.4. Case Scenarios

Three different case scenarios are discussed below for optimized round robin scheduling algorithm.

3.4.4.1 Best Case Scenario [Complexity]

The best case scenario of optimized round robin scheduling algorithm is when the sorting process is performed only once in the queue. In this case time complexity of this algorithm comes out to be O(nlogn).

Optimized round robin scheduling algorithm shows better results as compared to round robin, deficit round robin and start time fair queuing algorithms in most of the cases. Sometimes round robin, deficit round robin and start time fair queuing algorithms might show a bit better result as compare to optimized round robin algorithm. But Performance difference among these algorithms in this case is not very large.

3.4.4.2. Worst Case Scenario [Complexity]

The worst case scenario of optimized round robin scheduling algorithm is when the sorting process is performed k times in the queue, where k is the number of processes in the queue. In this case time complexity of this algorithm comes out to be O(k * nlogn).

Optimized round robin scheduling algorithm shows better results as compared to round robin, deficit round robin and start time fair queuing algorithms.

3.4.4.3. Average Case Scenario [Complexity]

The average case scenario of optimized round robin scheduling algorithm is when the sorting process is performed between minimum i.e. I and maximum value i.e. k (number of processes) in the queue. In this case time complexity of this algorithm comes out to be O(m*nlogn) where m>1 and m<k. Optimized round robin scheduling algorithm shows better results as compared to round robin, deficit round robin and start time fair queuing algorithms.

CHAPTER 4

TESTING AND PERFORMANCE EVALUATION

4.1. Introduction

The performance of the newly developed algorithm has been discussed in this chapter. To check the newly designed algorithm's performance and also to compare it with the performance of round robin, deficit round robin and start time fair queuing scheduling algorithms these four scheduling algorithms have been applied to different process sets. 10 sets of processes, with each set containing 10 processes, have been executed with these four algorithms.

4.2. Performance

To check the performance of optimized round robin scheduling algorithm 10 sets of processes with each set containing 10 processes, are executed with these four algorithms and then waiting and turnaround times for every process have been calculated. The execution of the processes is depicted via Gantt charts. Then average waiting and turnaround times values are calculated. These values are compared for these algorithms. This has shown performance of each algorithm i.e. optimized round robin, round robin, deficit round robin and start time fair queuing scheduling algorithm.

In each set of processes a value is selected for time quantum. Time quantum value should be carefully selected because if this value is very small too much context switching occurs and this waists CPU time. This CPU time can be used in some other computation. Therefore too much context switching should be avoided by selecting large time quantum value. But on the other hand if time quantum is very large then this would give CPU to large processes for more time and waiting and turn around times for small processes present after the large processes increase very much.

Therefore time quantum value is selected such that 80 percent of processes complete their execution when each of them get the CPU time for a maximum of one time quantum. [22]

Following are the sets of processes that are executed with these algorithms and then their results are also listed to show their comparison.

4.2.1. Process Set 1

Time Quantum = 25 ms

Process	Arrival Time	Burst Time
P0	0	30
P1	0	26
P2	0	23
P3	1	20
P4	2	19
P5	5	18
P6	9	10
P7	9	3
P8	10	2
P9	10	1

Detailed Calculation

Waiting Time = Sum of times spent by a process in ready queue from the time of its submission to the time of its completion

Turn Around Time = Waiting Time + Burst Time

1. Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P0	P1	
0	25	50	73	93	112	130	140	143	145	146	151	152

Waiting Times

P0 = 121	P1 = 126	P2 = 50	P3 = 72	P4 = 91
P5 = 107	P6 = 121	P7 = 131	P8 = 133	P9 = 135

Turn Around Times

P0 = 151	P1 = 152	P2 = 73	P3 = 92	P4 = 110
P5 = 125	P6 = 131	P7 = 134	P8 = 135	P9 = 136

2. Gantt chart Optimized Round Robin

P2	P9	P8	P7	P6	P0	P0	P5	P4	P1	P1	P3	
0	23	24	26	29	39	64	69	87	106	131	132	152

Waiting Times

P0 = 39	P1 = 106	P2 = 0	P3 = 131	P4 = 85
P5 = 64	P6 = 20	P7 = 17	P8 = 14	P9 = 13

Turn Around Times

P0 = 69	P1 = 132	P2 = 23	P3 = 151	P4 = 104
P5 = 82	P6 = 30	P7 = 20	P8 = 16	P9 = 14

3. Gantt chart Deficit Round Robin

P2	P1	P3	P4	P5	P6	P7	P8	P9	P0	
0	23	49	69	88	106	116	119	121	122	152

Waiting Times

P0 = 122	P1 = 23	P2 = 0	P3 = 48	P4 = 67
P5 = 83	P6 = 97	P7 = 107	P8 = 109	P9 = 111

Turn Around Times

P0 = 152	P1 = 49	P2 = 23	P3 = 68	P4 = 86
P5 = 101	P6 = 107	P7 = 110	P8 = 111	P9 = 112

4. Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	30	56	79	99	118	136	146	149	151	152

Waiting Times

P0 = 0	P1 = 30	P2 = 56	P3 = 78	P4 = 97
P5 = 113	P6 = 127	P7 = 137	P8 = 139	P9 = 141

Turn Around Times

P0 = 30	P1 = 56	P2 = 79	P3 = 98	P4 = 116
P5 = 131	P6 = 137	P7 = 140	P8 = 141	P9 = 142

	Optimized Round Robin	**Round Robin**	**Deficit Round Robin**	**Start Time Fair Queuing**
Aver. Waiting Time	48.9	108.7	76.7	91.8
Aver. Turn around Time	64.1	123.9	91.9	107.0

The processes are in descending order i.e. larger processes are present at the start of the queue and smaller processes are present at the end of the queue. The Gantt charts for optimized round robin, round robin, deficit round robin and start time fair queuing algorithms are shown above. The waiting and turn around times are calculated for each process. The average waiting and turn around times are calculated and have been shown in a table. The value for time quantum is selected as 25 milliseconds i.e. 80 percent of the processes can complete their execution in one time quantum.

Optimized round robin scheduling algorithm shows much better results as compared to round robin, deficit round robin and start time fair queuing algorithms i.e. optimized round robin scheduling algorithm shows smaller average waiting and turn around times values for the processes as compared to the other algorithms. This is because of the fact that in optimized round robin scheduling algorithm sorting process is performed. This decreases the waiting and turn around times for the smaller processes which in turn decrease average waiting and turn around times values for the processes. Process P9 which is present at the end of the queue requires only 1 millisecond for its execution. Process P9 has a waiting time of 13 ms for optimized round robin algorithm which is much less than the other algorithms' waiting time values.

In round robin, deficit round robin and start time fair queuing algorithms sorting process is not performed. The waiting and turn around times for the small processes present at the end of the queue are increased. This results in increasing the average waiting and turn around times for the processes.

4.2.2. Process Set 2

Time Quantum = 25 ms

Process	Arrival Time	Burst Time
P0	0	2
P1	1	2
P2	1	3
P3	1	3
P4	1	4
P5	5	10
P6	8	15
P7	12	22
P8	15	28
P9	15	30

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P8	P9

0 2 4 7 10 14 24 39 61 86 111 114 119

Gantt chart Optimized Round Robin

P0	P1	P4	P2	P3	P5	P6	P9	P9	P7	P8	P8

0 2 4 8 11 14 24 39 64 69 91 116 119

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9

0 2 4 7 10 14 24 39 61 89 119

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9

0 2 4 7 10 14 24 39 61 89 119

	Optimized Round Robin	**Round Robin**	**Deficit Round Robin**	**Start Time Fair Queuing**
Aver. Waiting Time	20.3	21.6	19.1	19.1
Aver. Turn around Time	32.2	33.5	31.0	31.0

In this process set processes are in ascending order i.e. small processes are present at the start of the queue and large processes are present at the end of the queue. The time quantum value is selected as 25 milliseconds.
In this case the difference between results is not very great. Although optimized round robin scheduling algorithm still shows better results as compared to round robin scheduling algorithm, the other two algorithms i.e. deficit round robin and start time fair queuing scheduling algorithms show a slight better results as compared to optimized round robin scheduling algorithm. But this difference in average waiting and turn around times is not great.

4.2.3. Process Set 3

Time Quantum = 15 ms

Process	Arrival Time	Burst Time
P0	0	5
P1	1	10
P2	1	7
P3	1	3
P4	5	10
P5	6	16
P6	8	2
P7	12	12
P8	12	8
P9	12	16

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P5	P9	
0	5	15	22	25	35	50	52	64	72	87	88	89

Gantt chart Optimized Round Robin

P0	P3	P6	P2	P9	P9	P8	P1	P5	P5	P4	P7	
0	5	8	10	17	32	33	41	51	66	67	77	89

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	5	15	22	25	35	51	53	65	73	89

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	5	15	22	25	35	51	53	65	73	89

	Optimized Round Robin	Round Robin	Deficit Round Robin	Start Time Fair Queuing
Aver. Waiting Time	25.1	32.0	28.6	28.6
Aver. Turn around Time	34.0	40.9	37.5	37.5

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 15 milliseconds. In this case optimized round robin scheduling algorithm shows better results i.e. less average waiting and turn around times as compared to round robin, deficit round robin and start time fair queuing.

4.2.4. Process Set 4

Time Quantum = 15 ms

Process	Arrival Time	Burst Time
P0	0	3
P1	0	2
P2	0	8
P3	0	2
P4	5	18
P5	9	24
P6	12	6
P7	20	12
P8	22	12
P9	25	2

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P4	P5	
0	3	5	13	15	30	45	51	63	75	77	80	89

Gantt chart Optimized Round Robin

P1	P3	P0	P2	P6	P7	P5	P9	P5	P8	P4	P4	
0	2	4	7	15	21	33	48	50	59	71	86	89

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	3	5	13	15	33	57	63	75	87	89

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	3	5	13	15	33	57	63	75	87	89

	Optimized Round Robin	**Round Robin**	**Deficit Round Robin**	**Start Time Fair Queuing**
Aver. Waiting Time	16.9	28.9	25.8	25.8
Aver. Turn around Time	25.8	37.8	34.7	34.7

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 15 milliseconds.

In this case process P9 requires only 2 milliseconds and is present at the end of the queue. Its waiting and turn around times increase for round robin, deficit round robin and start time fair queuing algorithms. Optimized round robin scheduling algorithm sorts the queue therefore waiting and turn around times for the process P9 are kept very low as compared to the other algorithms. Therefore optimized round robin scheduling algorithm shows better results i.e. less average waiting and turn around times as compared to round robin, deficit round robin and start time fair queuing.

4.2.5. Process Set 5

Time Quantum = 15

Process	Arrival Time	Burst Time
P0	0	10
P1	1	16
P2	1	15
P3	1	5
P4	2	2
P5	8	13
P6	10	16
P7	10	12
P8	10	10
P9	11	5

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P1	P6	
0	10	25	40	45	47	60	75	87	97	102	103	104

Gantt chart Optimized Round Robin

P0	P4	P3	P6	P6	P9	P8	P1	P1	P7	P5	P2	
0	10	12	17	32	33	38	48	63	64	76	89	104

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	10	26	41	46	48	61	77	89	99	104

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	10	26	41	46	48	61	77	89	99	104

	Optimized Round Robin	**Round Robin**	**Deficit Round Robin**	**Start Time Fair Queuing**
Aver. Waiting Time	33.3	53.7	44.3	44.3
Aver. Turn around Time	43.7	64.1	54.7	54.7

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 15 milliseconds.

Again in this case average waiting and turn around times of the processes for optimized round robin scheduling algorithm are quite less than the average waiting and turn around times of the processes for round robin, deficit round robin and start time fair queuing. This is because of the fact that optimized round robin scheduling algorithm performs the process of sorting. The rest of the three scheduling algorithms do not perform this sorting process. Therefore the average values for waiting and turn around times are increased for these algorithms.

4.2.6. Process Set 6

Time Quantum = 10 ms

Process	Arrival Time	Burst Time
P0	0	8
P1	1	3
P2	1	7
P3	1	6
P4	1	5
P5	2	2
P6	8	4
P7	10	1
P8	10	13
P9	11	22

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P8	P9	P9	
0	8	11	18	24	29	31	35	36	46	56	59	69	71

Gantt chart Optimized Round Robin

P0	P2	P7	P5	P1	P6	P4	P3	P8	P9	P8	P9	P9	
0	8	15	16	18	21	25	30	36	46	56	59	69	71

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	8	11	18	24	29	31	35	36	49	71

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	8	11	18	24	29	31	35	36	49	71

	Optimized Round Robin	**Round Robin**	**Deficit Round Robin**	**Start Time Fair Queuing**
Aver, Waiting Time	18.3	20.6	19.6	19.6
Aver. Turn around Time	25.4	27.7	26.7	26.7

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 10 milliseconds.
Again in this case average waiting and turn around times of the processes for optimized round robin scheduling algorithm are quite less than the average waiting and turn around times of the processes for round robin, deficit round robin and start time fair queuing.

4.2.7. Process Set 7

Time Quantum = 30 ms

Process	Arrival Time	Burst Time
P0	0	5
P1	1	12
P2	5	23
P3	5	27
P4	5	31
P5	9	65
P6	10	29
P7	20	20
P8	22	19
P9	22	5

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P4	P5	P5	
0	5	17	40	67	97	127	156	176	195	200	201	231	236

Gantt chart Optimized Round Robin

P0	P1	P2	P9	P8	P7	P5	P3	P6	P5	P5	P4	P4	
0	5	17	40	45	64	84	114	141	170	200	205	235	236

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	5	17	40	67	98	163	192	212	231	236

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	5	17	40	67	98	163	192	212	231	236

	Optimized Round Robin	Round Robin	Deficit Round Robin	Start Time Fair Queuing
Aver. Waiting Time	67.2	95.8	94.3	94.3
Aver. Turn around Time	90.8	119.4	117.9	117.9

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 30 milliseconds.
Again in this case average waiting and turn around times of the processes for optimized round robin scheduling algorithm are quite less than the average waiting and turn around times of the processes for round robin, deficit round robin and start time fair queuing. The difference between average waiting and turn around times is very significant in this case. This is because of the fact that in this process set most of the processes are large and only two of the processes are small. Therefore waiting and turn around times for the small processes are increased greatly in round robin, deficit round robin and start time fair queuing. On the other hand in case of optimized round robin scheduling algorithm as sorting is performed therefore waiting and turn around times for the small processes are kept very small plus waiting and turn around times for large processes are also kept as small as possible.

4.2.8. Process Set 8

Time Quantum = 10 ms

Process	Arrival Time	Burst Time
P0	0	8
P1	0	6
P2	7	12
P3	8	3
P4	9	1
P5	11	19
P6	20	9
P7	20	3
P8	20	4
P9	20	1

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P2	P5	
0	8	14	24	27	28	38	47	50	54	55	57	66

Gantt chart Optimized Round Robin

P1	P0	P4	P3	P2	P5	P9	P2	P7	P8	P6	P5	
0	6	14	15	18	28	38	39	41	44	48	57	66

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	8	14	26	29	30	49	58	61	65	66

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	8	14	26	29	30	49	58	61	65	66

	Optimized Round Robin	**Round Robin**	**Deficit Round Robin**	**Start Time Fair Queuing**
Aver. Waiting Time	16.7	22.5	22.5	22.5
Aver. Turn around Time	23.3	29.1	29.1	29.1

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 10 milliseconds.
Again in this case average waiting and turn around times of the processes for optimized round robin scheduling algorithm are quite less than the average waiting and turn around times of the processes for round robin, deficit round robin and start time fair queuing.
The difference between average waiting and turn around times is significant in this case. In this process set most of the processes are small and only two of the processes are large. Therefore waiting and turn around times for the small processes present at the end of the queue are increased greatly in round robin, deficit round robin and start time fair queuing. On the other hand in case of optimized round robin scheduling algorithm as sorting is performed therefore waiting and turn around times for the small processes are kept very small plus waiting and turn around times for large processes are also kept as small as possible. Therefore the average waiting and turn around times for the optimized round robin scheduling algorithm are smaller as compared to average waiting and turn around times for the rest of the algorithms.

4.2.9. Process Set 9

Time Quantum = 30 ms

Process	Arrival Time	Burst Time
P0	0	39
P1	1	24
P2	4	30
P3	4	21
P4	4	19
P5	10	35
P6	11	25
P7	15	29
P8	22	22
P9	28	20

Gantt chart Round Robin

P0	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P0**	**P5**	
0	30	54	84	105	124	154	179	208	230	250	259	264

Gantt chart Optimized Round Robin

P0	**P0**	**P5**	**P5**	**P4**	**P9**	**P2**	**P3**	**P8**	**P7**	**P1**	**P6**	
0	30	39	69	74	93	113	143	164	186	215	239	264

Gantt chart Deficit Round Robin

P0	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P0**	
0	30	54	84	105	124	159	184	213	235	255	264

Gantt chart Start Time Fair Queuing

P0	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	
0	39	63	93	114	133	168	193	222	244	264

	Optimized Round Robin	**Round Robin**	**Deficit Round Robin**	**Start Time Fair Queuing**
Aver. Waiting Time	116.7	139.4	131.4	117.0
Aver. Turn around Time	143.1	168.8	158.8	143.4

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 30 milliseconds.
All the processes in this process set are large. Again in this case average waiting and turn around times of the processes for optimized round robin scheduling algorithm are quite less than the average waiting and turn around times of the processes for round robin and deficit round robin algorithms. Start time fair queuing algorithm shows better results in this case which are quite close to optimized round robin algorithm but not better than optimized round robin algorithm.

4.2.10. Process Set 10

Time Quantum = 20 ms

Process	Arrival Time	Burst Time
P0	0	9
P1	0	8
P2	0	2
P3	0	12
P4	0	25
P5	0	4
P6	0	11
P7	0	9
P8	0	29
P9	0	1

Gantt chart Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P4	P8	
0	9	17	19	31	51	55	66	75	95	96	101	110

Gantt chart Optimized Round Robin

P9	P2	P5	P1	P0	P7	P8	P8	P6	P3	P4	P4	
0	1	3	7	15	24	33	53	62	73	85	105	110

Gantt chart Deficit Round Robin

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	
0	9	17	19	31	56	60	71	80	109	110

Gantt chart Start Time Fair Queuing

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9

0 9 17 19 31 56 60 71 80 109 110

	Optimized Round Robin	Round Robin	Deficit Round Robin	Start Time Fair Queuing
Aver. Waiting Time	30.3	46.9	45.2	45.2
Aver. Turn around Time	41.3	57.9	56.2	56.2

In this process set processes are neither in ascending nor in descending order i.e. small and large processes are present randomly in the queue without any order. The time quantum value is selected as 20 milliseconds.

This process set contains both small and large processes. Again in this case average waiting and turn around times of the processes for optimized round robin scheduling algorithm are quite less than the average waiting and turn around times of the processes for round robin, deficit round robin and start time fair queuing algorithms.

4.3 Performance Evaluation of Optimized Round Robin

For judging the performance of optimized round robin algorithm against round robin, deficit round robin and start time fair queuing algorithms, graphs have been drawn in the following section. Graphs have been drawn for waiting times and turn around times of the process sets listed above.

4.3.1. Graph for Waiting Time

To check the performance of optimized round robin scheduling algorithm graph have been drawn for waiting times of the process sets listed above. As there are 10 sets listed above therefore graph have been drawn for these sets of processes being executed with optimized round robin, round robin, deficit round robin and start time fair queuing algorithms.

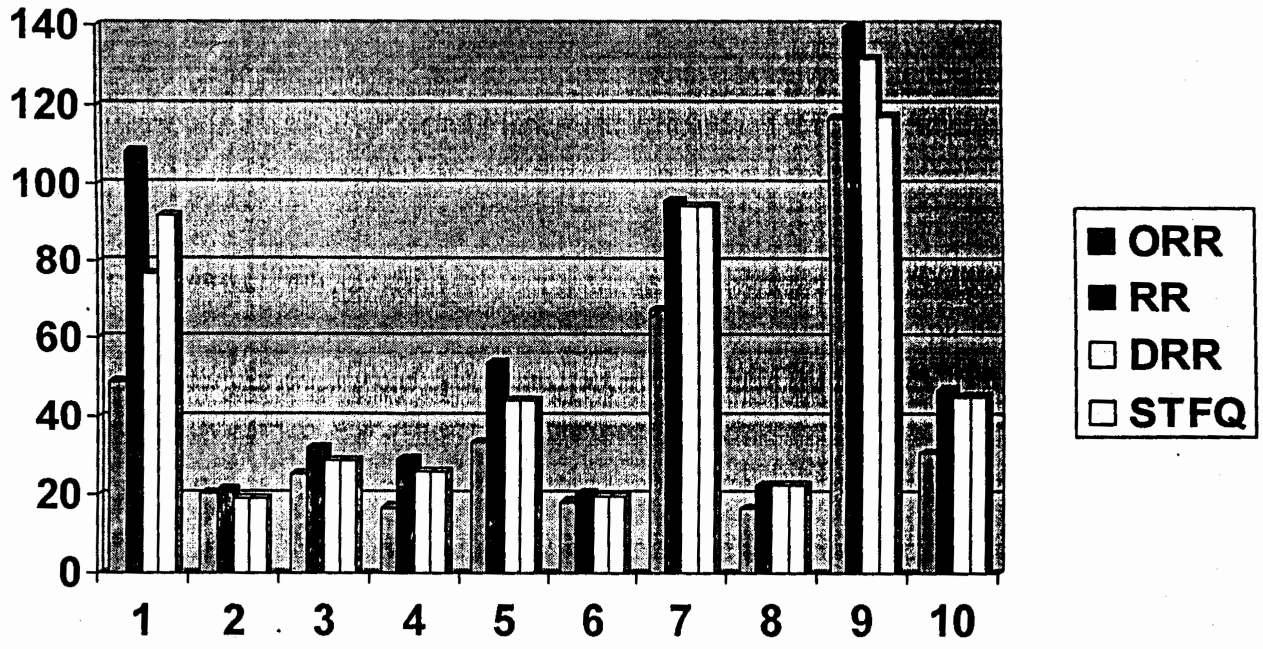

Figure 5.1 Graph showing waiting times of 10 sets

ORR	= Optimized Round Robin	RR	= Round Robin
DRR	= Deficit Round Robin	STFQ	= Start Time Fair Queuing

The graph clearly shows that optimized round robin scheduling algorithm is showing better results as compared to other scheduling algorithms. In each case optimized round robin scheduling algorithm is exhibiting less waiting time as compared to other scheduling algorithms. This means that optimized round robin scheduling algorithm is more optimized and efficient as compared to the algorithms whose results are shown above for waiting time criterion.

4.3.2. Graph for Turn around Time

To check the performance of optimized round robin scheduling algorithm graph have been drawn for turn around times of the process sets listed above. As there are 10 sets listed above therefore graph have been drawn for these sets of processes being executed with optimized round robin, round robin, deficit round robin and start time fair queuing algorithms.

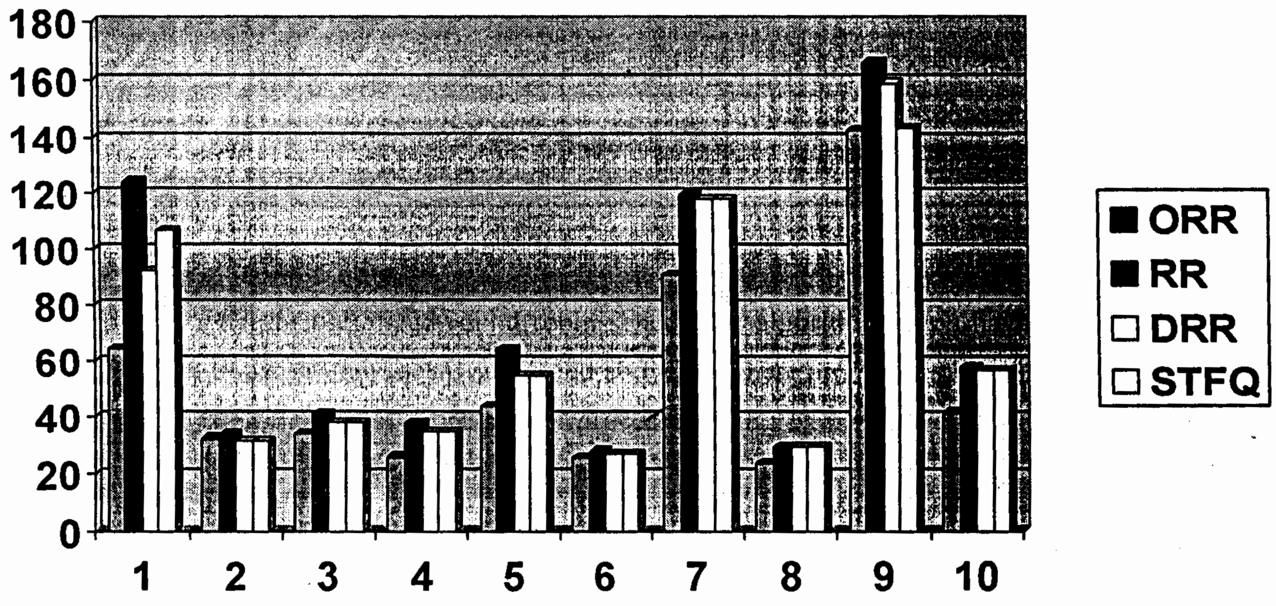

Figure 5.2 Graph showing turn around times of 10 sets

ORR	= Optimized Round Robin	RR	= Round Robin
DRR	= Deficit Round Robin	STFQ	= Start Time Fair Queuing

The graph clearly shows that optimized round robin scheduling algorithm has shown better results as compared to other scheduling algorithms. In each case optimized round robin scheduling algorithm is exhibiting less turn around time as compared to other scheduling algorithms. This means that optimized round robin scheduling algorithm is more optimized and efficient as compared to the algorithms whose results are shown above for turn around time criterion too.

4.3.3. Overall Performance

At the end let's take the average of average waiting time and average turn around time of all the ten sets. From these ten sets following results are calculated.

1- FOR OPTIMIZED ROUND ROBIN

Average Waiting Time

Average Waiting Time = [Av. Waiting time (set 1) + Av. Waiting time (set 2) + Av. Waiting time (set 3) + ... + Av. Waiting time (set 10)] / 10

Average Waiting Time = [48.9 + 20.3 + 25.1 + 16.9 + 33.3 +18.3 + 67.2 + 16.7 + 116.7 + 30.3] / 10

Average Waiting Time = 393.4 / 10

Average Waiting Time = 39.34

Average Turn around Time

Average Turn around Time = [Av. TA time (set 1) + Av. TA time (set 2) + Av. TA time (set 3) + ... + Av. TA time (set 10)] / 10

Average Turn around Time = [64.1 + 32.2 + 34.0 + 25.8 + 43.7 +25.4 + 90.8 + 23.3 + 143.1 + 41.3] / 10

Average Turn around Time = 523.7 / 10

Average Turn around Time = 52.37

2- FOR ROUND ROBIN

Average Waiting Time

Average Waiting Time = [Av. Waiting time (set 1) + Av. Waiting time (set 2) + Av. Waiting time (set 3) + ... + Av. Waiting time (set 10)] / 10

Average Waiting Time = [108.7 + 21.6 + 32.0 + 28.9 + 53.7 +20.6 + 95.8 + 22.5 + 139.4 + 46.9] / 10

Average Waiting Time = 569.9 / 10

Average Waiting Time = 56.99

Average Turn around Time

Average Turn around Time = [Av. TA time (set 1) + Av. TA time (set 2) + Av. TA time (set 3) + ... + Av. TA time (set 10)] / 10

Average Turn around Time = [123.9 + 33.5 + 40.9 + 37.8 + 64.1 +27.7 + 119.4 + 29.1 + 165.8 + 57.9] / 10

Average Turn around Time = 700.1 / 10

Average Turn around Time = 70.01

3- FOR DEFICIT ROUND ROBIN

Average Waiting Time

Average Waiting Time = [Av. Waiting time (set 1) + Av. Waiting time (set 2) + Av. Waiting time (set 3) + ... + Av. Waiting time (set 10)] / 10

Average Waiting Time = [76.7 + 19.1 + 28.6 + 25.8 + 44.3 + 19.6 + 94.3 + 22.5 + 131.4 + 45.2] / 10

Average Waiting Time = 507.5 / 10

Average Waiting Time = 50.75

Average Turn around Time

Average Turn around Time = [Av. TA time (set 1) + Av. TA time (set 2) + Av. TA time (set 3) + ... + Av. TA time (set 10)] / 10

Average Turn around Time = [91.9 + 31.0 + 37.5 + 34.7 + 54.7 + 26.7 + 117.9 + 29.1 + 158.8 + 56.2] / 10

Average Turn around Time = 638.5 / 10

Average Turn around Time = 63.85

4- FOR START TIME FAIR QUEUING

Average Waiting Time

Average Waiting Time = [Av. Waiting time (set 1) + Av. Waiting time (set 2) + Av. Waiting time (set 3) + ... + Av. Waiting time (set 10)] / 10

Average Waiting Time = [91.8 + 19.1 + 28.6 + 25.8 + 44.3 + 19.6 + 94.3 + 22.5 + 117.0 + 45.2] / 10

Average Waiting Time = 508.2 / 10

Average Waiting Time = 50.82

Average Turn around Time

Average Turn around Time = [Av. TA time (set 1) + Av. TA time (set 2) + Av. TA time (set 3) + ... + Av. TA time (set 10)] / 10

Average Turn around Time = [107.0 + 31.0 + 37.5 + 34.7 + 54.7 + 26.7 + 117.9 + 29.1 + 143.4 + 56.2] / 10

Average Turn around Time = 638.2 / 10

Average Turn around Time = 63.82

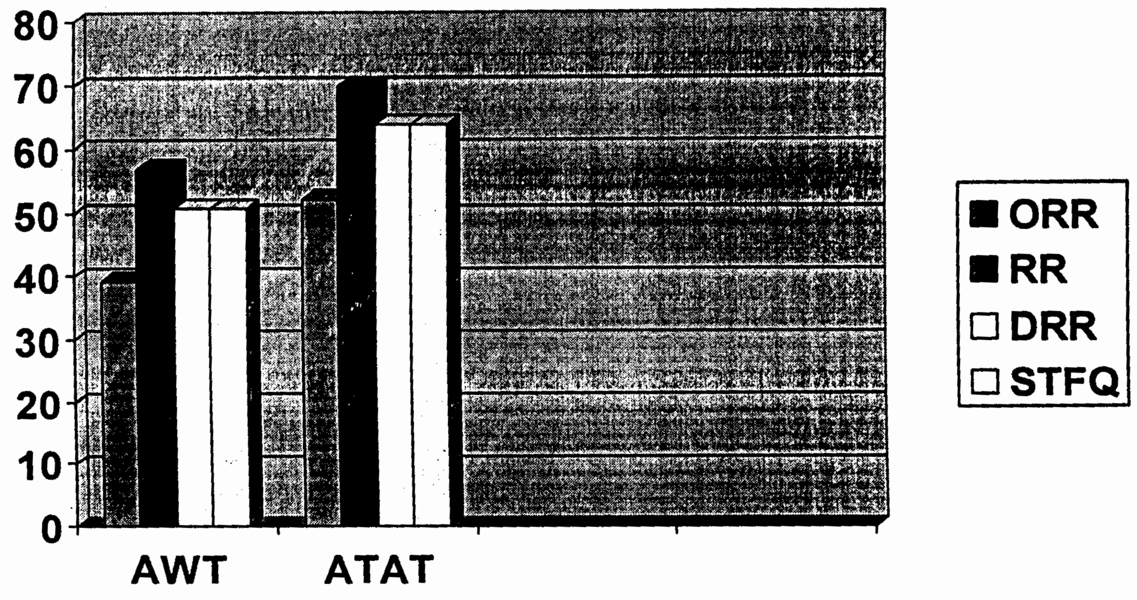

Figure 5.3 Graph showing overall performance

ORR	= Optimized Round Robin
RR	= Round Robin
DRR	= Deficit Round Robin
STFQ	= Start Time Fair Queuing
AWT	= Average Waiting Time
ATAT	= Average Turn Around Time

The above graph clearly shows the performance of optimized round robin scheduling algorithm against round robin, deficit round robin and start time fair queuing scheduling algorithms. The graph clearly shows that optimized round robin scheduling algorithm has shown better results i.e. low waiting and turn around times as compared to the other algorithms.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Introduction

Overall conclusion of this dissertation is discussed in this chapter. It also discusses the future work in this direction so that this algorithm can be made more efficient and optimized.

5.2. Conclusion

This dissertation discusses in detail the round robin scheduling algorithm. This dissertation was chosen because round robin scheduling algorithm is very widely used in different fields computer science including operating system, network devices etc. But this classical round robin scheduling algorithm has a problem associated with it that has been discussed in detail in this dissertation.

This dissertation focuses on the specific problem associated with round robin scheduling algorithm in detail and then proposes a new scheduling algorithm named as optimized round robin scheduling algorithm which resolves this issue. By resolving this issue optimized round robin scheduling algorithm has become more useful and effective.

The results of this new scheduling algorithm have been compared with results of round robin (RR), deficit round robin (DRR) and start time fair queuing (STFQ) scheduling algorithms and it has become obvious that optimized round robin (ORR) scheduling algorithm exhibits more optimized and efficient results as compare to RR, DRR and STFQ scheduling algorithms. Waiting and turn around times are the two criteria on which the results of these algorithms i.e. ORR, RR, DRR and STFQ scheduling algorithms have been tested. Results show that optimized round robin scheduling algorithm gives more optimized results for these two criteria than RR, DRR and STFQ scheduling algorithms.

From these results different graphs have been drawn to show the performance difference between optimized round robin scheduling algorithm and other scheduling algorithms. Hence it has become obvious and clear that optimized round robin scheduling algorithm gives more optimized results as compared to RR, DRR and STFQ scheduling algorithms.

5.3. Future Work

As it is said,

"There is always room at the top".

Therefore this proposed scheduling algorithm is not the last one. More optimized results can be obtained by applying some other variation of round robin scheduling algorithm.

Optimized round robin scheduling algorithm has shown better results compared to other algorithms discussed in this dissertation but optimized round robin scheduling algorithm does this at the cost of sorting. In optimized round robin sorting could occur frequently which consumes CPU time and this is overhead of optimized round robin scheduling algorithm that we have to bear.

One thing that can be done with this scheduling algorithm is that the ready queue is sorted many times if some sort of technique can be applied to this queue and sorting occurrence can be minimized this would also be a very useful future work related to this scheduling algorithm.

REFERENCES:

[1] Bennet J. and Zhang H., Hierarchical packet fair queueing algorithms, In ACM SIGCOMM '96 (1996).

[2] Chan W., and Nieh J., Group ratio round-robin: An O(1) proportional share scheduler, Tech. Rep. CUCS-012-03, Department of Computer Science, Columbia University, April 2003.

[3] Cheung S., and Pencea C., BSFQ: Bin sort fair queuing, In IEEE INFOCOM'02 (2002).

[4] Chuanxiong G. SRR, an O(1) time complexity packet scheduler for flows in multi-service packet networks, In ACM SIGCOMM '01 (2001).

[5] Clark D., and Fang W., Explicit allocation of best-effort packet delivery service, IEEE/ACM Transactions on Networking 6 (August 1998).

[6] D. Saha, M. Saksena, S. Mukherjee and S. Tripathi, On Guaranteed Delivery of Time-Critical Messages in DQDB, In Proc, IEEE Infocomm '94.

[7] D. Stiliadis and A. Varma, Efficient fair queueing algorithms for packet-switched networks, IEEE/ACM Trans. Networking 6 (2) (1998) 175-185.

[8] D. Stiliadis and A. Varma, Latency-rate servers: A general model for analysis of traffic scheduling algorithms, IEEE/ACM Trans. Networking 6 (5) (1998) 611-624.

[9] Goyal P., and Vin H., Generalized guaranteed rate scheduling algorithms: A framework, IEEE/ACM Transactions on Networking 5 (August 1997).

[10] H.M. Chaskar and U. Madhow, Fair scheduling with tunable latency: A Round Robin approach, in: IEEE Globecom'99, 1999, pp. 1328-1333.

[11] J.C.R. Bennett, D.C. Stephens and H. Zhang, High speed, scalable, and accurate implementation of fair queuing algorithms in ATM networks, in: Proceedings of the ICNP'97, 1997, pp. 7-14.

[12] J.L. Rexford, A.G. Greenberg and F.G. Bonomi, Hardware-efficient fair queueing architectures for high-speed networks, in: Proceedings of the INFOCOM '96, 1996, pp. 638-646.

[13] Lenzini L., Mingozzi E. and Stea G. Aliquem: a novel DRR implementation to achieve better latency and fairness at O(1) complexity, In IWQoS'02 (2002).

[14] M. Katevenis, S. Sidiropoulos and C. Courcoubetis, Weighted Round-Robin cell multiplexing in a general-purpose ATM switch chip, IEEE J. Selected Areas Commun. 9 (8) (1991) 1265-1279.

[15] M. Shreedhar and George Varghese, Efficient Fair Queuing using Deficit Round Robin by SIGCOMM '95 Cambridge, MA USA 1995 ACM 0-89791 -711-1 /95/0008

[16] Nick McKeown, The iSLIP Scheduling Algorithm for Input-Queued Switches, IEEE/ACM TRANSACTIONS ON NETWORKING VOL. 2, NO. 5, OCTOBER 1994

[17] N. Matsufuru and R. Aibara, Efficient fair queueing for ATM networks using Uniform Round Robin, in: Proceedings of the INFOCOM'99, 1999, pp. 389-397.

[18] Pawan Goyal, Harrick M. Vin, and Haichen Cheng, Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks, IEEE/ACM Transactions on Networking (1997) 690 – 704.

[19] Richard O. LaMaire and Dimitrios N.Serpanos, Two-Dimensional Round-Robin Scheduler for packet switches with multiple input queues, 0634692/94 1994 IEEE

[20] S. Golestani, A self clocked fair queueing scheme for broadband applications, In Proc. IEEE Infocomm '94, 1994.

[21] Shih-Chiang Tsao and Ying-Dar Lin, Pre-order Deficit Round Robin: a new scheduling algorithm for packet-switched networks, Computer Networks 35 (2001) 287-305

[22] Silberschatz, Galvin and Gagne, Operating System Concepts Windows XP updated with Java 7th edition

[23] S.J. Golestani, A self-clocked fair queueing scheme for broadband applications, in: Proceedings of the INFOCOM'94, April 1994, pp. 636-646.

[24] Sriram Ramabhadran and Joseph Pasquale, Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay, SIGCOMM '03, August 25-29, 2003, Karlsruhe, Germany ACM 1-58113-735-4/03/0008

[25] S. Suri, G. Varghese and G. Chandranmenon, Leap forward virtual clock: A new fair queuing scheme with guaranteed delays and throughput fairness, in: Proceedings of the INFOCOM'97, 1997, pp. 557-562.

[26] Stephens D., Bennet J., and Zhang H., Implementing scheduling algorithms in high speed networks, IEEE Journal on Selected Areas in Communications: Special Issue on Next-generation IP Switches and Routers 17 (June 1999).

[27] Stiliadis, D. and Varma A., Rate proportional servers: A design methodology for fair queueing algorithms, IEEE/ACM Transactions on Networking 6 (April 1998).

[28] www.wikipedia.org

[29] Xu, J. and Lipton R., On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms, In ACM SIGCOMM '02 (2002).

[30] Yao-Tzung Wung, Tzung-Puo Lin and Kuo-Chung Gan, An Improved Scheduling Algorithm for Weighted Round-Robin Cell Multiplexing in an ATM Switch, Computer and Communication Research Lab. Industrial Technology Research Institute, Taiwan. 0-7803-1825-0/94 1994 IEEE.