CHAPTER- ONE

INTRODUCTION

1.1 Introduction

The performance and efficiency of multitasking operating systems mainly depends upon the use of CPU scheduling algorithm. In time shared system, round robin (RR) scheduling gives optimal solution because due to its fairness and starvation free nature towards the processes, which is achieved by using the time quantum. As time quantum is static, it causes less context switching in case of high time quantum and high context switches in case of less time quantum. So, the performance of the system solely depends upon choice of the optimal time quantum which is dynamic in nature [6]. Overall performance of the system depends on the choice of an optimum time quantum, so the context switching can be reduced. Increasing context switches leads to high overhead and degrades the system performance, which results into high average waiting time and high average turnaround time. There are varieties of techniques to make quantum dynamic[1,2,5,6,9,10,11,12,17]. One of the widely researched approach is "Dynamic time quantum round robin scheduling algorithm. The idea of this approach is to make the time quantum dynamic based on time Q. The time Q for one complete round robin of all active process is to be kept constant, then quantum q can be computed dynamically after every Q seconds (or cycle) to reduce the mentioned performance parameters.

1.2 Motivation

Modern operating systems become more complex they have evolved from a single task to a multitasking environment in which process run in a concurrent manner. CPU scheduling is an essential operating system task; therefore its scheduling is central to operating system design. When there is more than one process in the ready queue waiting its turn to be assigned to the CPU, the operating system much decide through the scheduler the order of execution [16]. Allocating CPU to a process requires careful attention to assures fairness and avoid process starvation for CPU. Scheduling decision try to minimize the following: average turnaround time, average waiting time, and number of context switches. The round robin algorithm is the main concern of this research which is the oldest, simplest, fairest and most widely used scheduling algorithm, designed especially for time-sharing systems. A small unit of time, called time slices

or quantum is defined. All runnable processes are kept in a circular queue. The CPU scheduler goes around this queue, allocating the CPU to each process for a time interval of one quantum. New processes are added to the tail of the queue. The CPU scheduler picks the first process from the queue, sets a timer to interrupt after one quantum and dispatches the process. If the process is still running at the end of the quantum, the CPU is preempted and the process is added to the tail of the queue [14]. If the process finishes before the end of the quantum, the process itself releases the CPU voluntarily. In either case, the CPU scheduler assigns the CPU to the next process in the ready queue. The performance of the RR algorithm depends heavily on the size of the time quantum [6, 12].At one extreme, if the time quantum is extremely large, it results into poor response time and approximates FCFS. On the other hand, if the time quantum is extremely small this causes too many context switches and lower the CPU efficiency. RR algorithm gives better responsiveness but worse average turnaround time and waiting time [10, 12,13,17]. A lot of attempts were developed to find a solution for the high turnaround time, high waiting time and the overhead of the extra context switches in round robin, regardless of the different methodologies used in these attempts; however all of these rely on the fixed time quantum. In last few years different approaches are used to increase the performance of round robin. SARR[12], DQRR[2], IRR[9], and TPBCS[3] based on dynamic time quantum was designed to solve all critical previously mentioned problems in a practical, simple and applicable manner.

1.2.1 Dynamic Quantum Approaches

If the time Q for one complete round robin of all active processes is to keep constant, then q (time quantum for a process) can be computed dynamically as:

$q = Q/n$

Here the relation between the quantum q and number of processes n is linear. This shows that the quantum or time slice as inverse relation with the number of jobs in ready queue.

Another method of achieving dynamic quantum is based on median [4]. This method calculates dynamic quantum by using the formula.

$$\text{Median(M)} = \begin{cases} Y(n+1)/2 \text{ if n is odd} \\ ½(\frac{Yn}{2} + Y\left(1 + \frac{n}{2}\right)) \text{ if n is even} \end{cases}$$

Where, M= Median

Y= number located in the middle of a group of numbers arranged in ascending order

n = number of processes

Then, q = (highest Bt+ median (M))/2

Where, Bt =burst time

q = quantum

The optimal time quantum is assigned to each process and is recalculated taking the remaining burst time in account after each cycle. This procedure goes on until the ready queue is empty.

As already stated large time quantum causes, results into poor response time and small quantum results into large context switch overhead and hence gives poor CPU efficiency. Thus one main goal of RR class schedulers is to keep the small quantum size and also minimize the number of context switches, which are conflicting requirement. This means increasing quantum size normally decreases context switches and vice versa. This dissertation work will propose an approach of making quantum dynamic which will neither make time slice fixed for one quantum nor will be based on burst time. Proposed approach will make quantum dynamic based on the fraction of jobs finished in previous round of round robin scheduling.

1.3 Performance metrics

Performance metrics refers the criteria for measuring the performance of any system. In the case of empirical evaluation of modified dynamic time quantum round robin scheduling algorithm context switch, average waiting time and average turnaround time are the key terms for measuring the performance. Lower the context switch, average waiting time and average turnaround time exhibits higher performance.

(a) Context switch

The number of times the CPU switches from one process to another is called the context switches. If context switches decreases, then throughput will increases.

(b) Turnaround time

This is the time difference of the arrival time and the finish time of the process. It is generally the sum of the waiting time and the service time of the process. If average turnaround time decreases, then throughput will increase.

$att = \Sigma(ft- at)/n$

where, att= average turnaround time

ft= finish time

at= arrival time

n= total number of jobs

(c) Waiting time

This is the amount of time that a process spends waiting on the ready queue. The waiting time should be kept minimum. Waiting time and throughput are directly dependent on each other. If average waiting time decreases, then it is clear that throughput will be increased.

$awt = \Sigma(tt- bt)/n$

Where, awt= average waiting time

tt= turnaround time

bt= burst time

(d) Average response time

Time from the submission of a request until the first response is produce.

So, a good scheduling algorithm for real time and time sharing system must possess following

1. Minimize context switch
2. Maximize CPU utilization
3. Minimize average turnaround time
4. Minimize average waiting time
5. Minimize response time

1.4 Problem statement

The performance of the round robin algorithm depends heavily on the size of the time quantum. If the time quantum is large, the round robin simply becomes FCFS and, if time quantum is small, there are so many preemptions of the CPU. Many context switches decrease the utilization of CPU because, in case of large number of context switch, CPU is busy with no fruitful work. Many attempts are done previously to make the quantum dynamic. This dissertation work will find answer to the question "What happens if we make the quantum dynamic by using: the fraction of jobs finished in previous round" as shown below:

$$\text{Quantum}(q) = \begin{cases} q * \frac{\lambda}{f} & \text{if } f \geq 0.0625 \\ 4q \text{ if } f \leq 0.0625 \end{cases}$$

Where, f= fraction of job finished in previous round

λ =0.25 (constant value)

1.5 Objective

The main objective of this dissertation work is:

(a). Experimental Evaluation of Modified Dynamic Time Quantum Round Robin Scheduling

(b). To compare Dynamic time quantum round robin scheduling and Modified dynamic time quantum round robin scheduling in terms of context switch, average waiting time and average turnaround time.

1.6 Thesis Organization

Introduction part of this dissertation work focuses on CPU scheduling algorithms and related basic terminologies which are already mentioned along with introduction to dynamic time quantum approach. This chapter also clarifies motivation and objectives of dissertation work. Chapter two consists of background and literature review. In background part include concept of multiprogramming, process model, CPU scheduler, categories of scheduling algorithms, stage of scheduler and static and dynamic RR scheduling algorithm are discussed. Literature review includes details of several dynamic round robin scheduling algorithms with their categories. Chapter three includes algorithms studied with their illustrations. Chapter four describes the implementation details which are used to implement this work such as tools, programming language, interface, data structure, algorithms and flowcharts. Chapter five explains the experimental analysis with three different cases. Chapter six concludes the whole dissertation work and recommends for further research.

CHAPTER TWO

BACKGROUND AND LITERATURE REVIEW

2.1 Background

An operating system is system software which makes an interface between user and hardware. Operating system provides a platform in which user can interact with hardware so that user can handle the system in an efficient manner [16]. Modern operating system and time sharing system are more complex, they have evolved from a single task to multitasking environment in which processes run in synchronized manner. In multiprocessing and multitasking environment there are several processes ready to run at the same time, the system must choose among them and assigned to run on the available CPUs is called CPU scheduling. In any multiprogramming system, the CPU switches from process to process quickly, running each for tens or hundreds of milliseconds. While strictly speaking, at any instance of time, the CPU is running only one process, in the course of one second, it may work on several of them, giving the illusion of parallelism called pseudo parallelism. Pseudo parallelism is a concept that contrasts with the true hardware parallelism of multiprocessor system (which has two or more CPUs sharing the physical memory). Keeping track of multiple, parallel activities is hard for people to do. Therefore, operating system designer over the years have involved a conceptual model (sequential processes) that make parallelism easier to deal with.

2.1.1 Multiprogramming

To overcome the problem of underutilization of CPU and main memory, the multiprogramming was introduced. The multiprogramming is interleaved execution of multiple jobs by the same computer. In multiprogramming system, when one program is waiting for I/O transfer; there is another program ready to utilize the CPU. So it is possible for several jobs to share the time of the CPU. But it is important to note that multiprogramming is not defined to be the execution of jobs at the same instance of time. Rather it does mean that there are a number of jobs available to the CPU (placed in main memory) and a portion of one is executed then a segment of another and so on. A simple process of multiprogramming is shown in figure below:

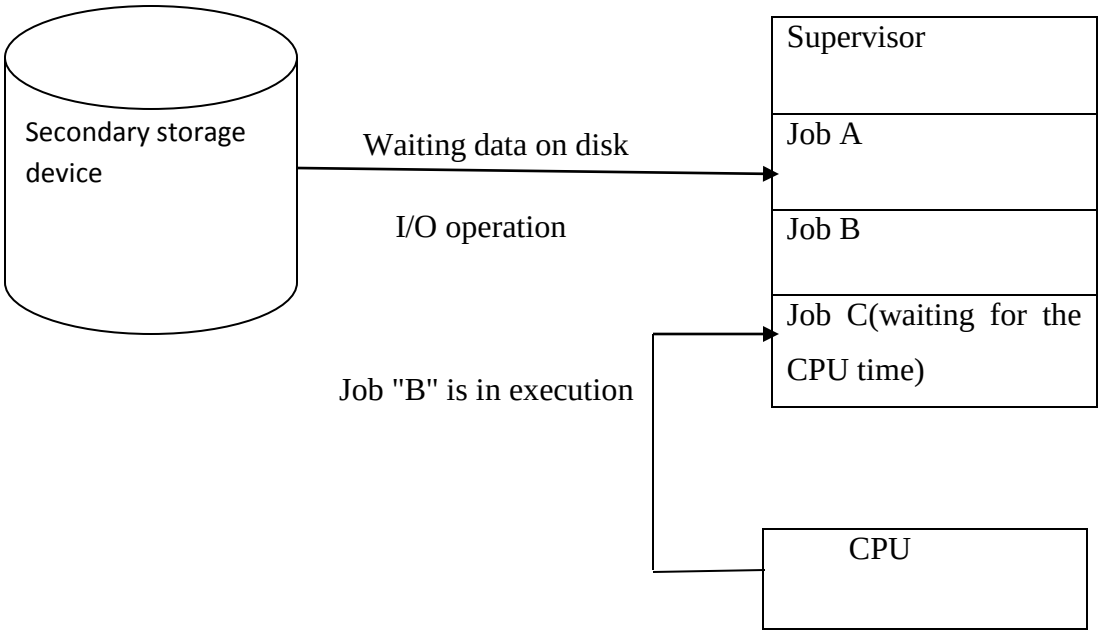

Figure 2.1: Memory layout for a multiprogramming system

As shown in figure, at the particular situation, job A is not utilizing the CPU time because it is busy in I/O operations. Hence the CPU becomes busy to execute the job B. Another job C is waiting for the CPU for getting its execution time. So in this state the CPU will never be idle and utilizes maximum of its time.

A program in execution is called a "process", "job" or a "task". The concurrent execution of programs improves the utilization of system resources and enhances the system throughput as compared to batch and serial processing. In this system, when a process requests some I/O to allocate; meanwhile the CPU time is assigned to another ready process. So here when a process is switched to an I/O operation, the CPU is not set idle.

Multiprogramming is a common approach to resource management. The essential components of a single user operating system include a command processor, an input/output control system, a file system, and a transient area. A multiprogramming operating system builds on this base, subdividing the transient area to hold several independent programs and adding resource management routines to the operating system's basic functions.

2.1.2 The Process Model

In this model, all runnable software on the computer, sometimes including the operating system, is organized into a number of sequential processes. A process is just an interface of an executing program, including the current values of the program counter, registers, and variables. Conceptually, each process has its own virtual CPU. In reality, of course, the real CPU switches back and forth from process to process[14,16], but to understand the system, it is much easier to think about a collection of processes running in (pseudo) parallel then to try to track of how the CPU switches from program to program.

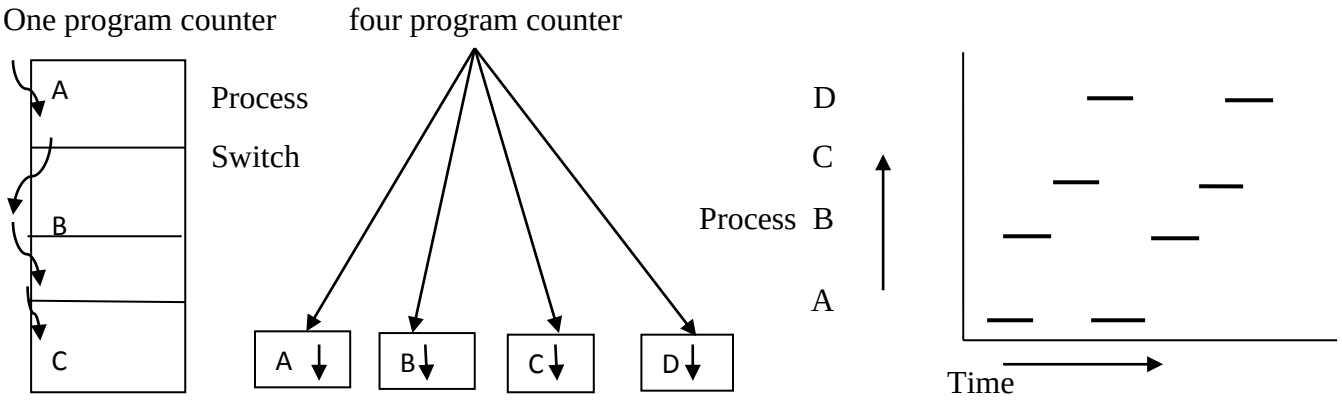

Figure 2.2: Process execution in pseudo- parallelism

2.1.3 Process State

When a process is loaded in memory, it becomes ready to execute. When the scheduler selects the process for execution, the process enters the running state. In this state, the process can either be preempted which is the case when it exceeds the time quantum allocated or blocked while waiting for I/O data. When process is preempted then the operating system puts the process on the end of the ready queue of processes, but it remains ready to execute. If the process is blocked while waiting for I/O operation, it is, then, taken from ready queue and put on the I/O queue. When I/O channel completes the I/O operation for blocked process, the process reenters the ready state, where it waits for CPU.

Thus, at any time each process may be in one of the following states:

New: The process being created

Ready: In this state, the process is ready to run, and waiting for CPU. This is the only state from where process can enter the running state.

Running: In this state, the process is using the CPU, and process can, either be preempted and put in the ready state, or may go to blocked state for I/O operation or may terminate with or without error. Simply, in this state instruction are being executed.

Blocked/Waiting: In this state, the process is waiting for I/O operation. When channel completes its I/O operation, then, the process becomes ready and the operating system puts it on the back of the ready queue.

Terminated: The process has finished execution.

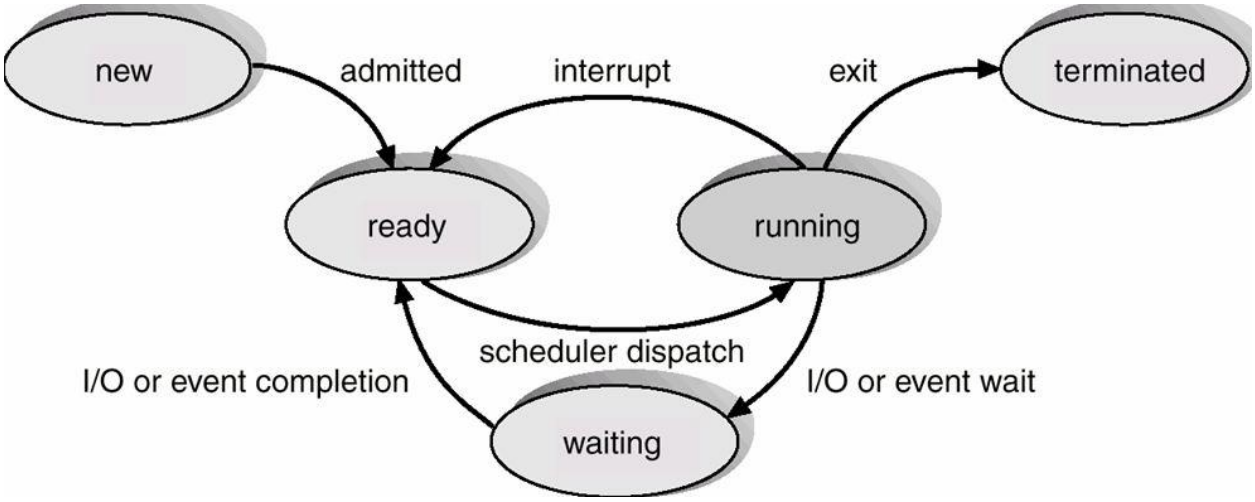

Figure 2.3: Process state diagram

2.1.4 CPU and I/O-bound processes

CPU-bound: CPU-bound means the rate at which process progresses is limited by the speed of the CPU. A task that performs calculations on a small set of numbers, for example multiplying small metrics, is likely to be CPU-bound.

I/O-bound: I/O-bound means the rate at which a process progresses is limited by the speed of the I/O subsystem. A task that processes data from disk, for example, counting the number of lines in a file is likely to be I/O-bound.

2.1.5 CPU Scheduling Algorithm

Scheduling is a fundamental operating system's function. CPU scheduling deals with the problem of deciding which of the process in the ready queue is to be selected for CPU. Thus, whenever the CPU becomes idle, the operating system must select from among the processes in memory that are ready to execute, and allocates the CPU to it. The part of the operating system which makes the choice as to which of the processes in the ready queue runs next is called scheduler, and the algorithm it uses is called scheduling algorithm.

Many CPU scheduling algorithms are used such as First come first served scheduling (FCFS), shortest job first scheduling (SJF), priority scheduling, etc. All the above algorithms are non-preemptive in nature and also not suitable for time sharing systems. In the first come first served (FCFS), the process that arrives first in the ready queue is allocated the CPU first. In SJF, when the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes have same next CPU burst time, FCFS scheduling is used to break the tie. In priority scheduling algorithm, a priority is given to each process and the process having highest priority is executed first and so on. Round robin scheduling is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a time quantum or time slice is defined and the CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of upto 1 time quantum. The round robin (RR) scheduling is one of the most popular scheduling algorithms found in computer systems today. In addition it is designed especially for time sharing systems and found in multiple processor system.

2.1.6 Categories of scheduling algorithms

Different scheduling algorithms are needed in different environment [16]. The objective of an operating system has different goals than the others. In other words, what the scheduler should optimize for is not the same in all systems, distinguishable three categories of operating systems, through some systems have mixtures of them, are:

(a)Batch systems

Batch processing is the execution of a series of programs (jobs) on a computer without manual intervention. Jobs are set up so they can be run to completion without human interaction. All input parameters are predefined through scripts, command line segments, control files or job control language. Non preemptive algorithms or preemptive algorithms with long time periods for each process are often acceptable. This approach reduces process switches and thus improves performance. The batch algorithms are actually fairly general and often applicable to other situations as well, which makes them worth studying, even for people not involved in corporate mainframe computing.

(b) Interactive system

In an interactive environment with interactive user, preemption is essential to keep one process from hogging the CPU and denying service to others. Even if no process intentionally runs forever, one might shut out all the others indefinitely due to a program bug. Preemption is needed to prevent this behavior also falls into this category, since they normally serve multiple (remote) users. All of whom are in a big hurry.

(c) Real time system

Real time systems detect and control the event outside the system under the timing constraints. If these timing constraints must be made to avoid catastrophe, the system is hard real time system otherwise non hard (soft) real time system. Real time systems add an extra dimension to the system namely time which makes them even harder to develop than other system. The difference with interactive system is that real time system run only programs that are intended to further the application at hand. Interactive systems are general purpose and run arbitrary program that are not cooperative or even malicious. Real time systems can be categorized into two categories: soft and hard real time system.

2.1.7 Stages of scheduler

A process migrates between the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from this queue in some fashion. The selection process is carried out by the appropriate scheduler [10]. Operating system may feature up to three distinct types of scheduler [14].

(a)Long term scheduling

Long- term scheduling performs a gate keeping function. It decides whether there is enough memory, or room, to allow new programs or jobs into the system. It limits the degree of multi-tasking to prevent slow performance on current running programs. When a job gets past the long term scheduler, it is sent on the medium- term scheduler.

(b)Medium term scheduling

The medium term scheduling makes the decision to send a job on or to sideline it until a more important process is finished. Later, when the computer is less busy or has less important jobs, the medium term scheduler allows the suspended job to pass.

(c)Short term scheduling

The short term scheduler takes jobs from the "ready" line and gives them the green light to run. It decides which of them can have resources and for how long. The short term scheduler runs the highest priority jobs first and must make on the spot decisions. For example, when a running process is interrupted and may be changed, the short term scheduler must recalibrate and give the highest priority job the green light.

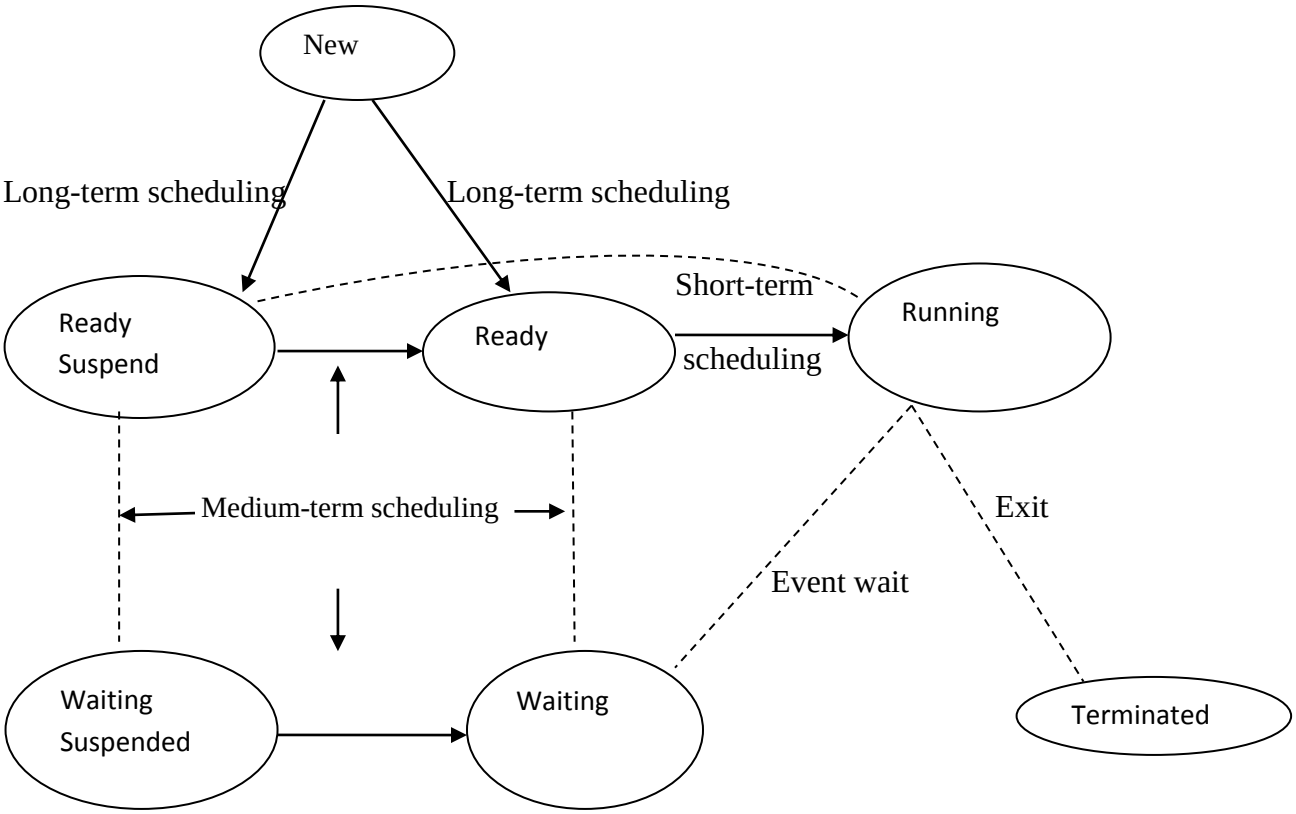

Figure 2.4: Queuing diagram for scheduling

All computer resources are scheduled before in use. So, CPU scheduling algorithm determines how the CPU will be allocated to the process. CPU scheduling algorithms are two types, one is non preemptive and another is preemptive scheduling algorithms [16]. In non-preemptive scheduling, once the CPU is allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. But, in preemptive scheduling, the CPU can release the processes even in the middle of the execution. A process is a program at the time of execution. A process is the unit of work in most systems. Such a system contains a collection of processes and it includes the program counter, the process stack, and the contents of process register, etc. Operating system processes execute system code, and user processes execute user code. All these processes may execute concurrently.

2.2 Literature Review

The evolution of scheduling closely tracked the development of computers. The concept of scheduling is not new; Hennery L. Gantt, an American engineer and social scientist is credited with the development of the bar chart (Gantt chart) in 1917 round robin scheduling algorithm. There are many variations of the primitive round robin scheduling algorithms.

2.2.1 FCFS scheduling algorithm

First come first served scheduling algorithm is one of the simplest non- preemptive scheduling algorithms. In this algorithm, the process that requests the CPU first gets the CPU first. The implementation of this algorithm consists of a FIFO queue of the ready processes. The process enters the ready queue and continuously moves to the front of the ready queue. When it reaches to the front of the queue, it is allocated the processor when it becomes free. This algorithm, generally, has long average waiting time. The main advantage of this algorithm is that it is easy to understand, easy to program, and ensures fairness.

2.2.2 Round robin scheduling algorithm

It is one of the most popular algorithms found in computer systems today for multiprogramming and time sharing environment. It is similar to FCFS, but preemption is included to switch the CPU among the processes. A time duration called quantum is introduced in this algorithm, it is the time for which CPU is assigned a process. Thus, each process is assigned the same time interval (time quantum) and, if the process exceeds its time quantum, CPU is preempted and is given to another process on the ready queue. The round robin scheduler has the advantage of very little selection overhead as scheduling is done in constant time. Thus, scheduling time is simply O(1) because it has to put running process to the end of the ready queue and has to select the process from the front of the queue, which takes the constant amount of time.

2.2.3 Weighted round robin (WRR)

The standard round robin does not deal with different priorities of processes. All processes are equally executed. In weighted round robin, quantum is based on the priorities of the processes. A high prioritized process receives a larger quantum, and by this, receives execution time

proportional with its priority. This is a very common extension to the primitive round robin scheduler and will be referred to simply as the round robin scheduler.

2.2.4 Virtual round robin (VRR)

The virtual round robin scheduler described by S. William [15] is an extension of the standard round robin scheduler. The round robin scheduler treat I/O bound processes and CPU- bound processes equally, but an I/O bound process does not fully use its time- slice and thus gets an unfair treatment compared to CPU- bound processes. The virtual round robin scheduler addresses the unfair treatment of I/O- bound processes by allowing processes to maintain their quantum when blocked, the quantum might be variable, and placing the blocked process at the front of the ready queue when it returns to the ready queue. A process is only returned to the back of the queue when it has used its full quantum. Researchers have shown that this algorithm is better than the standard round robin scheduler in terms of fairness between I/O bound processes and CPU-bound processes.

2.2.5 Virtual time round robin (VTRR)

The weighted round robin and virtual round robin schedulers both use a variable quantum for processes, as priorities are implemented by changing the quantum given to each processes. In the virtual time round robin N. Jason and T. Andrew [7] use a fixed quantum, but change the frequency by which a process is executed in order to implement priorities. This has the advantage that response times are generally improved for high prioritized processes, while the selection overhead is still constant time.

2.2.6 Self adjusted round robin scheduling algorithm (SARR)

The static time quantum which is a limitation of RR was removed by taking dynamic time quantum using median method introduced in SARR[12] algorithm. SARR algorithm is based on a new approach called dynamic time quantum in which, the quantum is repeatedly adjusted according to the burst time of the running processes.

2.2.7 Dynamic quantum with re-adjusted round robin scheduling algorithm (DQRRR)

The DQRRR [2] scheduling has improved the RR scheduling by improving the turnaround time, waiting time and number of context switches. Processes are arranged in job mix order in the ready queue and time quantum is found using median method. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. Again the time quantum is calculated from the remaining burst time of the processes and so on. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue and allocates the CPU to the process for 1 time quantum. DQRRR gives better result than classical RR scheduling algorithm. According to[2], from a list of N processes, the process which needs minimum CPU time is assigned the time quantum first and the highest from the list and so on till the Nth process.

2.2.8 Improved round robin scheduling algorithm (IRR)

In IRR [9] algorithm, the processes are arranged in ascending order according to their burst time present in the ready queue. Optimal time quantum is calculated by following the median method. The optimal time quantum is assigned to each process and is recalculated taking the remaining burst time in account after each cycle. This procedure goes on until the ready queue is empty. This is better than DQRRR and RR. Here all the processes are CPU bound. No processes are I/O bound.

2.2.9 A New proposed two processor based CPU scheduling algorithm with Varying time quantum for real time system (TPBCS)

TPBCS[3] algorithm finds the time quantum in an intelligent way which gives better result in a two- processor environment than dynamic quantum with readjusted round robin scheduling algorithm(DQRRR) and RR scheduling. Out of the two processors one is solely dedicated to execute CPU-intensive processes (CPU1) and the other CPU is solely dedicated to execute I/O-intensive processes (CPU2). The algorithm is divided into part1 and part2. Part1 algorithm classifies a process and dispatches it into an appropriate ready queue. Part2 algorithm calculates the time quantum for both CPUs in a dynamic manner in each cycle of execution. The time quantum is repeatedly adjusted in every round, according to the remaining burst of the currently

running processes. We have taken an approach to get the optimal time quantum, where a percentage value <PCcpu, PCi/o> is assigned to each processes.

2.2.10 Average max round robin algorithm (AMRR)

P. Banerjee, S. S. Dhal, [5] developed a scheduling algorithm called Average Max Round Robin algorithm (AMRR). Here time quantum is the mean of the summation of the average and maximum burst time.

2.2.11 Min-Max Round Robin (MMRR):

S. K. Panda, S. K. Bhoi, [11] developed a scheduling algorithm called Min Max Round Robin (MMRR). In this approach time quantum is taken as the range of the CPU burst time of all the processes. The range of the processes is the difference between largest (maximum) and smallest (minimum) values.

2.2.12 Comparative performance analysis of Multi-Dynamic time quantum round robin (MDTQRR) algorithm with arrival time

Behera et al. [1] also proposed a method called Multi Dynamic Time Quantum Round Robin (MDTQRR) which dynamically calculated the value of the time quantum and leads to increase in system throughput.

2.2.13 Finding Time Quantum of Round Robin CPU Scheduling in general computing systems using integer programming

S. M. Mostafa, S. Z. Rida, Hamad, [8] finding time quantum of Round Robin CPU Scheduling algorithm in general computing system using integer programming. In this paper integer programming has been proposed to solve equations that decide a value that is neither too large nor too small such that every process has reasonable response time and the throughput of the system is not decreased due to unnecessary context switches. This method developed a changing time quantum in each round over the cyclic queue. This method is called as Changeable Time Quantum (CTQ) technique

CHAPTER THREE

ALGORITHMS STUDIED

3.1 Algorithm Study Framework

To describe the working of algorithms, Dynamic time quantum round robin scheduling and Modified dynamic time quantum RR scheduling through hand tracing, a set of random number of processes is taken (here five processes are taken). The algorithm works efficiently even if it used with a very large number of processes. All the experiments are performed for uniprocessor environment and the processes taken are CPU bound processes only. Here input process is considered independent from each other and arrival time of each process taken is considered zero. Time slice is assumed to be not more than the maximum burst time.

In this hand traced experiment, several input and output parameters are considered. The input parameters consist of burst time, time quantum and number of processes. The output parameter consists of average waiting time (awt), average turnaround time (att), and number of context switch (cs).

3.2 Studied Algorithm

3.2.1 Dynamic Time Quantum RR scheduling

This scheduling algorithm, identify the impact on overall performance of batch MOS by Varying quantum time over fixed time quantum round robin scheduling. The time quantum q may be constant for long period of time or it may vary with process load [7]. If the time Q for one complete round robin of all active processes is to be kept constant, then q can be computed dynamically every Q seconds (or cycle) as:

q =Q/n

Here the relationship between the quantum q and number of processes n is linear i.e. the quantum or time slice as inverse relationship with the number of jobs in the ready queue. Consequently, whenever there are fewer jobs in the queue, jobs are privileged with larger quantum thereby

reducing the context switches. Inversely, whenever there are larger numbers of jobs, each job will get small but optimal time slice maintaining the optimal result.

Subsequently,

(a) att = Σ(ft- at)/n

Where, att= average turnaround time

ft= finish time

at= arrival time

n= total number of jobs

(b) awt = Σ(tt- bt)/n

where, awt= average waiting time

tt= turnaround time

bt= burst time

(c) The scheduling algorithm runs in constant time known as O(1), regardless of number of jobs in the system.

3.2.2 Modified Dynamic Time Quantum Round Robin (MDTRR) Scheduling Algorithm

In Dynamic time quantum round robin scheduling, the value of Q may be constant in all round while calculating quantum q. In modified algorithm, first of all the value of quantum q is given fixed. After that we calculate the quantum q dynamically by using modified formula as:

$$\text{Quantum (q)} = \begin{cases} q * \frac{\lambda}{f} & \text{if } f \geq 0.0625 \\ 4q & \text{if } f \leq 0.0625 \end{cases}$$

Where, f= fraction of job finished in previous round

λ =0.25 (constant value)

Here, we assume the value of λ fixed which is 0.25 and then calculating the f which is fraction of jobs finished in previous round. The value of f is calculated as:

f =number of jobs finished in previous round/total number of jobs in that round

After calculating f, we check the value of f with 0.0625. if f is greater than or equal to 6.25% then the value of quantum q is minimized but not less than four times which is given by formula **q= q* λ/f.** In some condition, instead of decreasing it increases the value to make the quantum optimum, and when value of f is less than 6.25% then the value of quantum q is maximized but not more than four times which is given by formula **q= 4q** and hence maintaining the optimal result.

3.3 Illustration

To demonstrate above mentioned algorithms(Dynamic time quantum round robin scheduling and modified dynamic time quantum round robin scheduling), arrival time is considered to be zero for the given processes P1, P2, P3, P4, P5 and corresponding burst time are random number. Burst time of these processes(P1, P2, P3, P4, P5) are 69, 35, 51, 52 and51 respectively shown in table number 3.1 and table number 3.2 shows the comparison result of DTRR and MDTRR. Gantt chart for each of these two algorithms is shown in figure 3.1 and figure 3.2

Number of processes	Burst time
P1	69
P2	35
P3	51
P4	52
P5	51

Table 3.1 Data in random order

First of all the value of Quantum 'Q' =100 and number of process 'n' is given. Here n=5.

DTRR

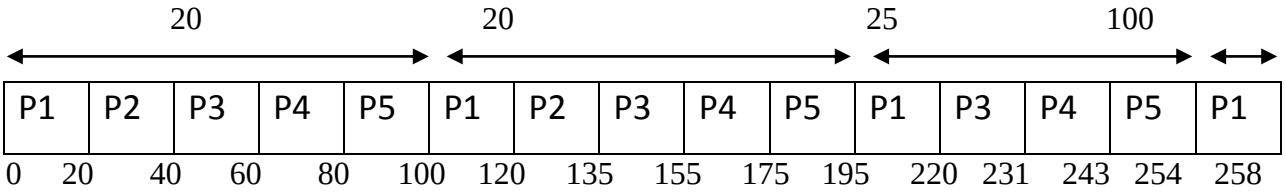

Figure 3.1: Gantt chart for DTRR

No of context switches = 14

Average waiting time = 132

Average turnaround time = 183

MDTRR

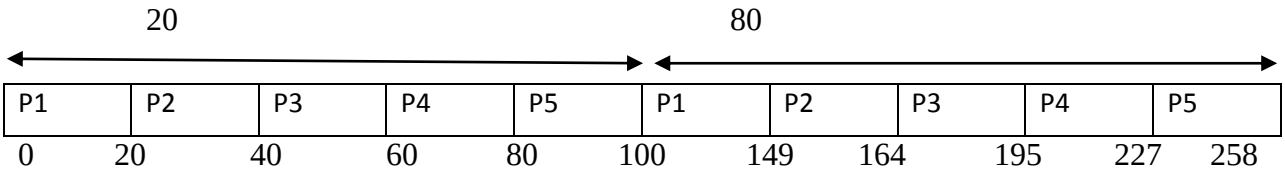

Figure 3.2: Gantt chart for MDTRR

Number of context switches = 9

Average waiting time = 107

Average turnaround time = 158

Accumulated View of Scheduler Algorithms

Algorithms	QT	CS	Awt	Atat
DTRR	20,20,25,100	14	132	183
MDTRR	20,80	9	107	158

Table 3.2: Comparison Table DTRR and MDTRR

CHAPTER FOUR

IMPLEMENTATION

4.1 Tools Used

4.1.1 Programming Language

C# programming language in .NET framework is used for simulating the DTRR and MDTRR algorithms. .NET is designed to provide an environment within which we can develop almost any application to run on windows, whereas C# is a programming language designed specially to work with the .NET framework. This simulator is developed on Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz 2.50GHz, 4GB RAM, window 8.1, 64-bit OS.

4.2 Data Structure Used

4.2.1 List

A list will be used to implement both the scheduling algorithms. Each node of list will contain different necessary data related to process. Lists are a way to store many different values under a single variable. Every item in this list is numbered with an index. By calling the list and passing it a particular index value, a programmer can pull out any item placed into it. The advantage of using list is that; linked list will usually take more memory than list because it needs space for all those next/previous references and the data will probably have less locality of reference, as each node is a separate object. The structure of list is illustrated in figure below:

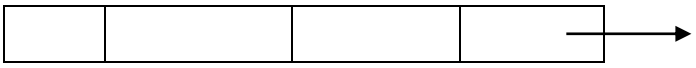

Figure 4.1: Structure of list

4.3 Algorithms and Flowcharts taken in this dissertation

4.3.1 Pseudo code for DTRR Scheduling Algorithm

1. I/P: Process (Pn), burst time(bt), arrival time(at), ready queue(rq)

   O/P: context switch (cs), average waiting time (awt), average turnaround time (att)

2. Initialize: ready queue=0, cs=0, awt=0, att=0, N= number of jobs in ready queue at a particular time, n= total number of jobs

3. for each process i

rbt[i]= bt[i]

4. Initialize the value of Quantum Q

5. Read Process in Ready Queue

6. While (ready queue!= EMP)

   6.1 q= Q/N

   6.2 for each process i

    If (rbt[i]<=q)

        Record finish time ft[i]

       Remove job from ready queue

       N--

     Else

        rbt[i]= rbt[i]-q

        cs++

    6.3 end for

7. end while

8. for each process i

    Calculate tat[i]=ft[i]-at[i]

    Calculate wt[i]= tat[i]-bt[i]

9. calculate Awt and Att

    Awt= ∑wt[i]/n

    Att= ∑tat[i]/n

10. stop and exit

4.3.2 Flowchart of DTRR Scheduling Algorithm

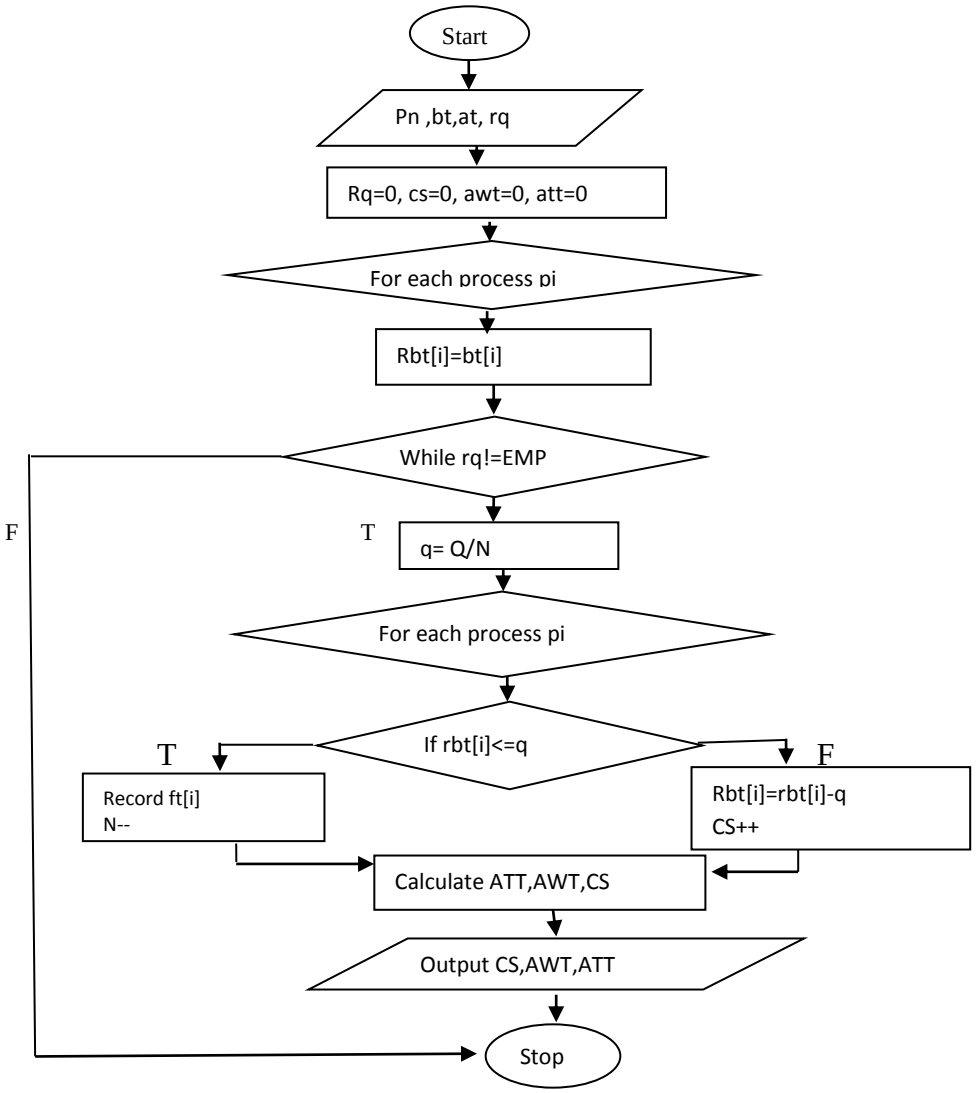

Figure 4.2: Flowchart of DTRR

4.3.3 Pseudo code for MDTRR Scheduling Algorithm

1. I/P: Process (Pn), burst time (bt), arrival time(at), ready queue(rq)

   O/P: context switch (cs), average waiting time (awt), average turnaround time (att)

2. Initialize: ready queue=0, cs=0, awt=0, att=0, λ= 0.25

3. for each process i

rbt[i]= bt[i]

4. While (ready queue!= EMP)

   4.1 for each process i

If (rbt[i]<=q)

    Record finish time ft[i]

    Remove job from ready queue

    N--

  Else

    rbt[i]= rbt[i]-q

    cs++

  4.2 end for

  4.3 calculate f= jobs finished in previous round/ total number of jobs in that round

  4.4 if (f>= 0.0625)

        q = q*λ/f

    else

q = 4q

5.end while

6. for each process i

Calculate tat[i]=ft[i]-at[i]

Calculate wt[i]= tat[i]-bt[i]

7. calculate Awt and Att

Awt= ∑wt[i]/n

Att= ∑tat[i]/n

8. stop and exit

4.3.4 Flowchart of MDTRR Scheduling Algorithm

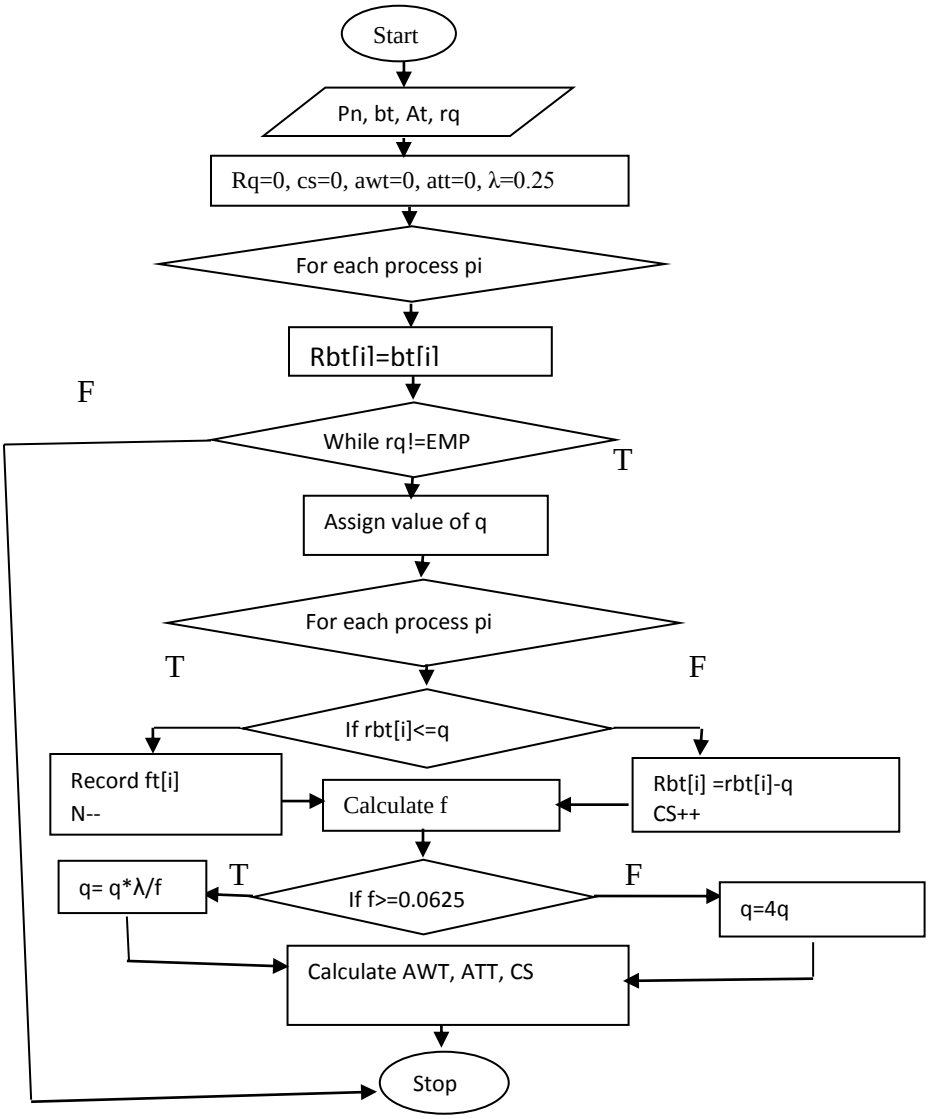

Figure 4.3: Flowchart of MDTRR

CHAPTER FIVE

DATA COLLECTION AND ANALYSIS

5.1 Test Case Design

To evaluate the algorithms taken in this dissertation, three different test cases are designed, in which each test case design is again categorized into three different cases such that the number of processes in each case is same but with different burst time ranges. In this way, the burst time range of each test case design as low, medium and high. Finally the algorithms taken are evaluated with three different test case design but taking the same parameters.

5.1.1 Test Case 1:

In this test case input number of processes are varies from 20 to 200 with interval size 20. Burst time of each process is generated using random number generator and the range of the burst time of these processes is taken in between 25 to 200 in data set 1, 200 to 500 in data set 2 and 500 to 1000 in data set 3.

5.1.2 Test Case 2:

In this test case input number of processes are varies from 50 to 500 with interval size 50. Burst time of each process is generated using random number generator and the range of the burst time of these processes is taken in between 200 to 500 in data set 1, 500 to 700 in data set 2 and 700 to 1000 in data set 3.

5.1.3 Test Case 3:

In this test case input number of processes are varies from 100 to 1000 with interval size 100. Burst time of each process is generated using random generator and the range of the burst time of these processes is taken in between 200 to 500 in data set 1, 500 to 700 in data set 2 and 700 to 1000 in data set 3.

Here, in all above three cases, the value of Q can be selected as: Q= N*10

5.2 Data Collection and Analysis

5.2.1 For Test case 1:

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
20	1676	1801	212	1619	1744	73
40	3380	3499	412	3151	3270	131
60	4301	4408	535	4104	4212	238
80	6195	6309	763	5808	5919	302
100	8994	9117	1078	8574	8697	341
120	10236	10355	1226	9895	10014	407
140	10210	10320	1255	10177	10287	561
160	12246	12357	1495	11860	11971	619
180	13770	13879	1678	13108	13217	583
200	14918	15027	1838	14494	14604	781

Table 5.1 Input processes are taken in 20 to 200 and their burst time ranges in between (25 to 200)

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
20	5572	5917	622	5248	5593	79
40	11513	11867	1258	10491	10845	157
60	18007	18365	1943	16431	16790	238
80	22557	22899	2436	20733	21074	317
100	29799	30158	3204	2681	27179	396
120	35549	35901	3797	32960	33312	476
140	42322	42682	4530	38334	38693	556
160	47375	47712	5061	42316	42670	631
180	51525	51868	5514	46892	47235	711
200	59397	59752	6321	54102	54456	795

Table 5.2 Input processes are taken in 20 to 200 and their burst time range in between (200 to 500)

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
20	13913	14716	1509	11906	12709	86
40	25725	26472	2725	20311	21058	169
60	25386	26133	2695	21372	22120	172
80	52588	53347	5511	42693	43452	346
100	65268	66009	6811	52946	53687	426
120	77385	78125	8047	62727	63467	513
140	92787	93543	9635	77921	78676	605
160	104808	105558	10866	88623	89373	688
180	118605	119352	12278	97859	98607	772
200	129607	130348	13453	105712	106453	854

Table 5.3 Input processes are taken in 20 to 200 and their burst time range in between (500 to 1000)

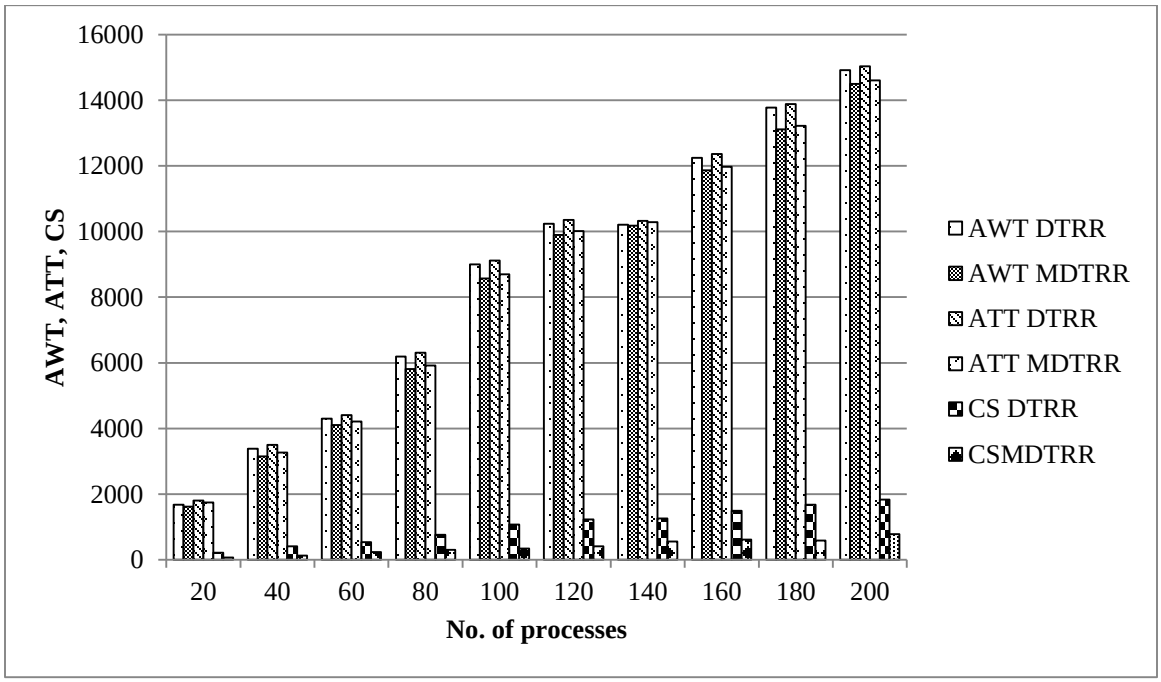

Figure 5.1: Graph for Table 5.1

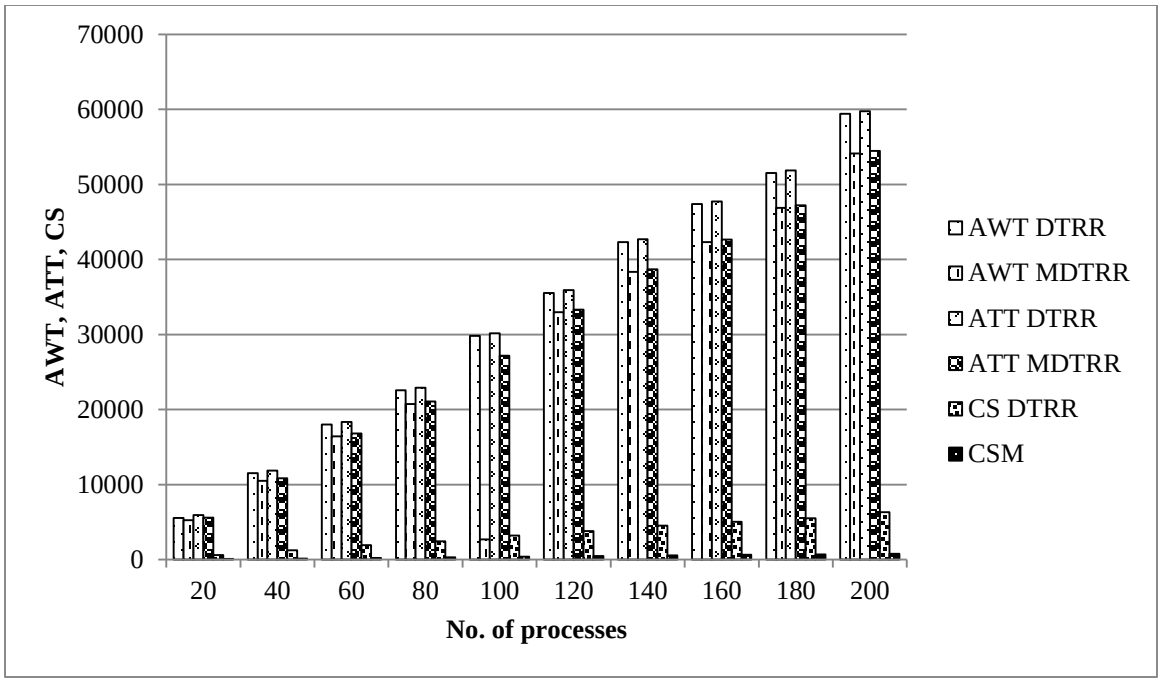

Figure 5.2: Graph for Table 5.2

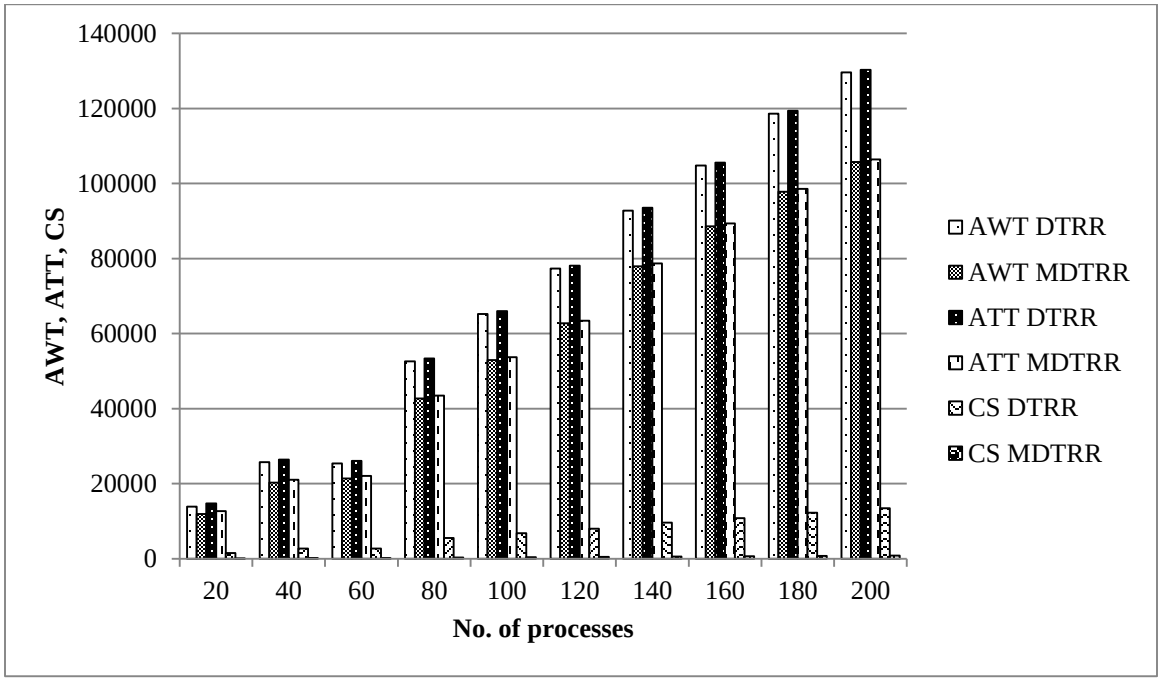

Figure 5.3: Graph for Table 5.3

If we look at case 1; in figure 5.1, the average waiting time of MDTRR is 3.52% to 4.89% lower and average turnaround time is 5.79% to 10.96% lower with respect to DTRR; in figure 5.2, the average waiting time of MDTRR is 6.17% to 11.11% lower and average turnaround time is 5.79% to 10.96% lower with respect to DTRR and in figure 5.3, , the average waiting time of MDTRR is 16.85% to 23.27% lower and average turnaround time is 15.79% to 22.44% lower with respect to DTRR. In all three figures, the context switch decreases drastically. In figure 5.3, AWT and ATT are decreased by greater percentage than in figure 5.1 and 5.2. It is only because of greater burst range value.

5.2.2 For Test Case 2:

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
50	13682	14019	1487	12599	12935	196
100	28725	29067	3083	26900	27242	398
150	45071	45428	4816	40342	40698	595
200	57987	58336	6190	53391	53740	791
250	75842	76200	8059	68654	69012	993
300	88173	88523	9389	81104	81453	1194
350	102620	102969	10941	93617	93966	1388
400	115144	115489	12233	105912	106257	1582
450	131900	132249	14033	121392	121741	1788
500	144874	145220	15387	133472	133819	1979

Table 5.4: Input processes are taken in 50 to 500 and their burst time range in between(200 to 500)

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
50	27127	27717	2855	19287	19877	199
100	56096	56701	5829	39979	40584	399
150	83798	84400	8684	59786	60388	599
200	111258	111853	11515	79316	79911	799
250	139324	139922	14409	99255	99853	999
300	168934	169536	17438	120251	120853	1199
350	196175	196776	20276	139674	140275	1399
400	223971	224570	23131	159324	159923	1599
450	252868	253469	26119	180371	180972	1799
500	278946	279542	28804	198543	199139	1999

Table 5.5: Input processes are taken in 50 to 500 and their burst time range in between(500 to 700)

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
50	38574	39421	4030	31492	32339	222
100	80153	81015	8288	69097	69959	456
150	118362	119211	12198	99191	100040	675
200	158782	159634	16324	133072	133925	900
250	200671	201528	20623	170395	171251	1136
300	239438	240294	24585	202967	203822	1361
350	276238	277086	28333	228696	229544	1569
400	318208	319059	32644	270235	271086	1800
450	355924	356770	36457	302197	303043	2026
500	397414	398267	40733	337263	338116	2254

Table 5.6: Input processes are taken in 50 to 500 and their burst time range in between(700 to 1000)

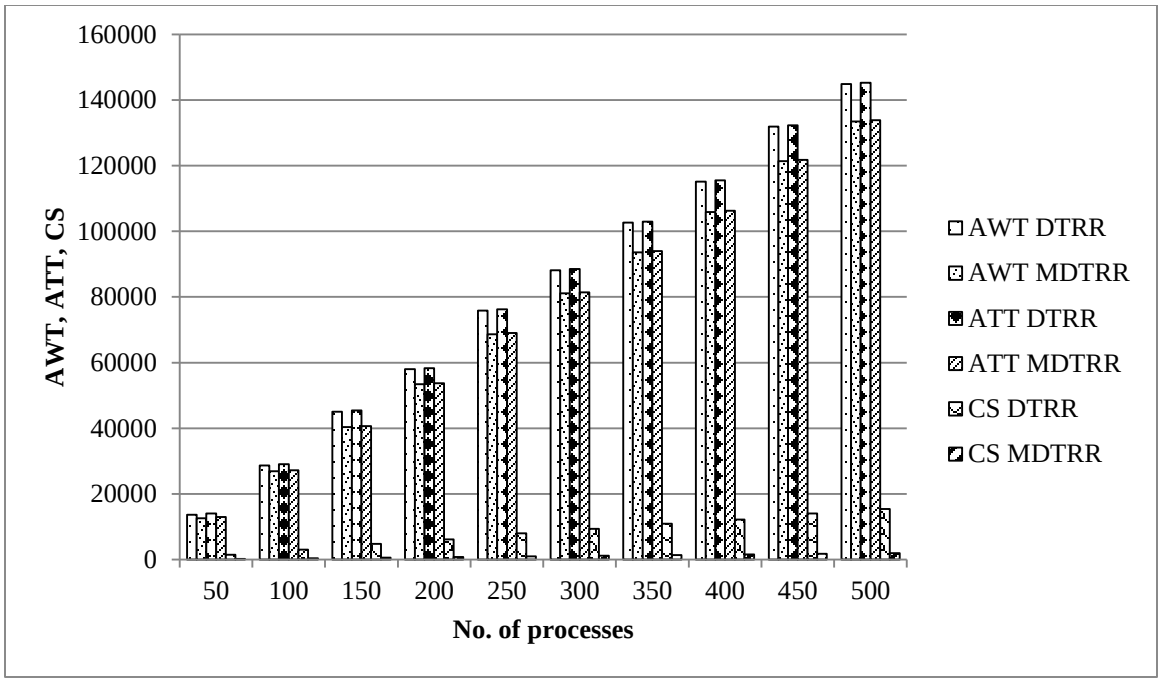

Figure 5.4: Graph for Table 5.4

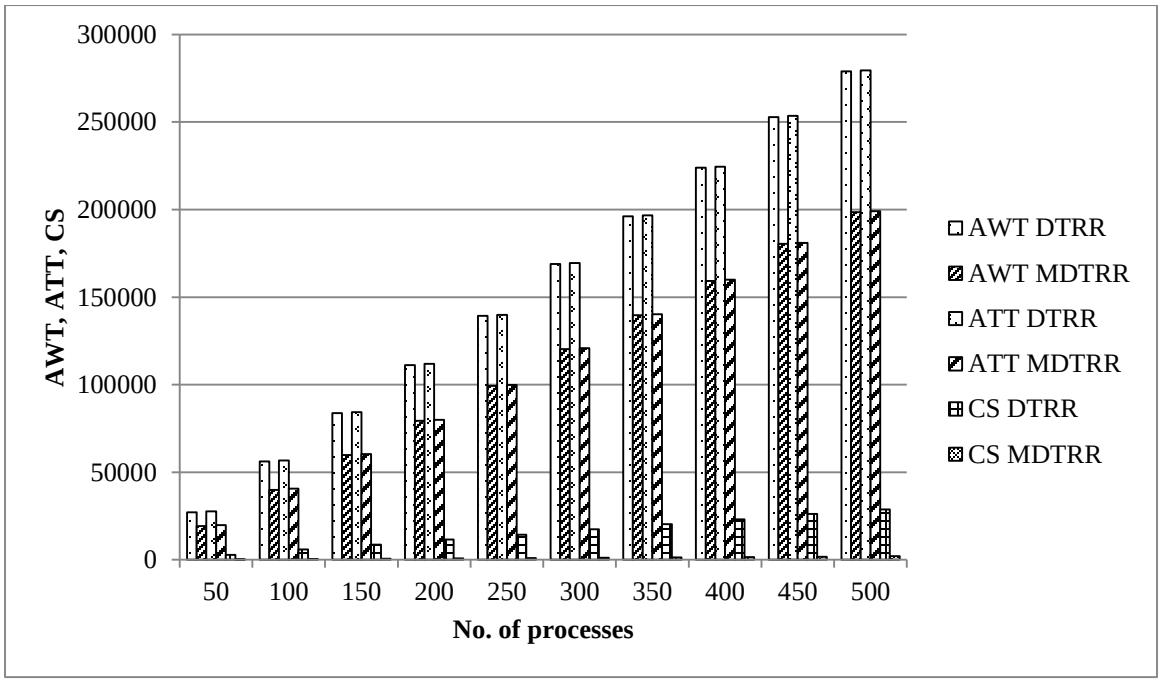

Figure 5.5: Graph for Table 5.5

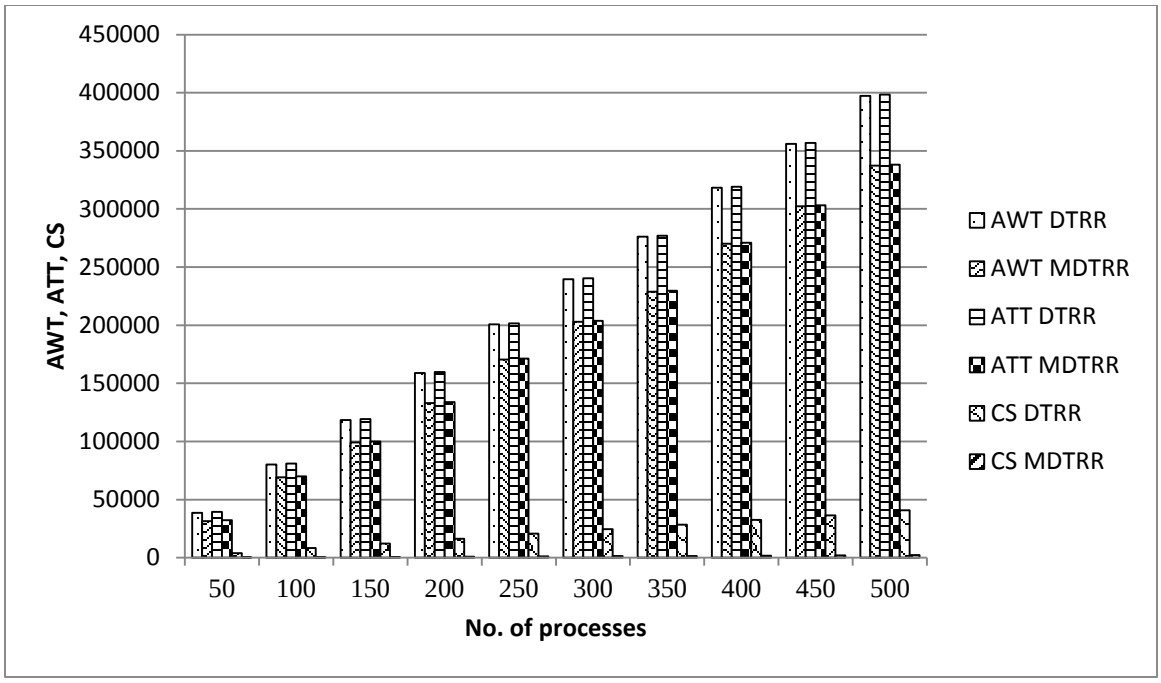

Figure 5.6: Graph for Table 5.6

If we look at case 2; in figure 5.4, the average waiting time of MDTRR is 8.54% to 10.46% lower and average turnaround time is 8.38% to 10.41% lower with respect to DTRR; in figure 5.5, the average waiting time of MDTRR is 40.36% to 40.64% lower and average turnaround time is 39.44% to 40.12% lower with respect to DTRR and in figure 5.6, the average waiting time of MDTRR is 17.76% to 22.48% lower and average turnaround time is 17.67% to 21.89% lower with respect to DTRR. In all three figures, the context switch decreases drastically. Here, in figure 5.5, AWT and ATT is decreased by greater percentage than in figure 5.4 and 5.6 because in some condition, the modified algorithm $q*\lambda/f$ doesn't decrease the value. Instead of decreasing it increases the value to make the quantum optimum and gives better result.

5.2.3 For Test Case 3

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
100	29056	29406	3116	25962	26312	393
200	60819	61178	6490	54925	55285	795
300	92515	92879	9856	83794	84159	1196
400	113505	113844	12099	104765	105104	1580
500	146038	146385	15530	134188	134535	1981
600	176511	176860	18727	161296	161646	2378
700	206805	207156	21931	188396	188747	2776
800	241712	242069	25690	218016	218373	3178
900	264964	265314	28151	242224	242573	3570
1000	297116	297468	31582	270444	270796	3969

Table 5.7: Input processes are taken in 100 to 1000 and their burst time range in between(200 to 500)

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
100	54262	54847	5647	38875	39461	399
200	111330	111929	11541	79363	79962	799
300	168018	168617	17366	119786	120385	1199
400	224822	225424	23219	159620	160222	1599
500	282520	283132	29166	200427	201031	1999
600	334586	335184	34483	238252	238849	2399
700	394642	395245	40675	280411	281014	2799
800	448077	448676	46210	318903	319501	3199
900	502187	502784	51674	358450	359047	3599
1000	559037	559635	57671	398772	399370	3999

Table 5.8: Input processes are taken in 100 to 1000 and their burst time range in between(500 to 700)

No of process	DTRR			MDTRR		
	AWT	ATT	CS	AWT	ATT	CS
100	78636	79486	8122	66794	67644	452
200	158387	159237	16277	132686	133536	900
300	237754	238602	24405	195665	196512	1343
400	314072	314915	32201	264159	265002	1792
500	395501	396350	40556	327889	328738	2248
600	475573	476421	48694	396040	396888	2697
700	552909	553756	56636	462785	463631	3135
800	635044	635895	65049	536453	537304	3609
900	715664	716514	73198	600950	601800	4043
1000	792793	793643	81201	661414	662264	4504

Table 5.9: Input processes are taken in 100 to 1000 and their burst time range in between(700 to 1000)

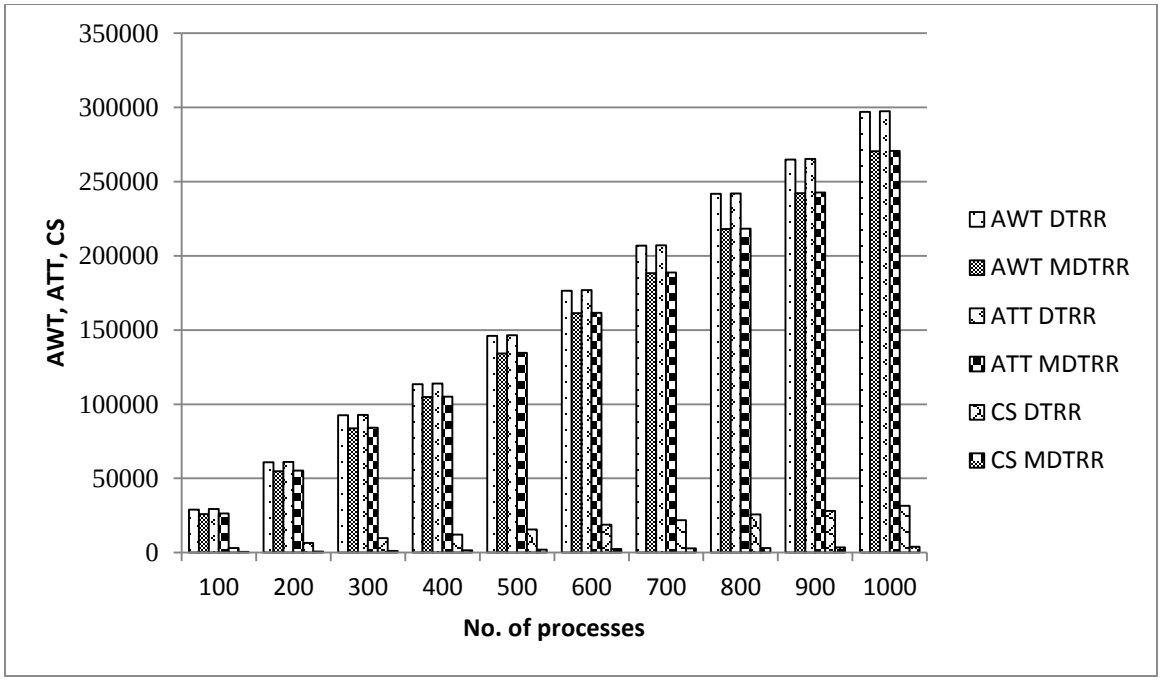

Figure 5.7: Graph for Table 5.7

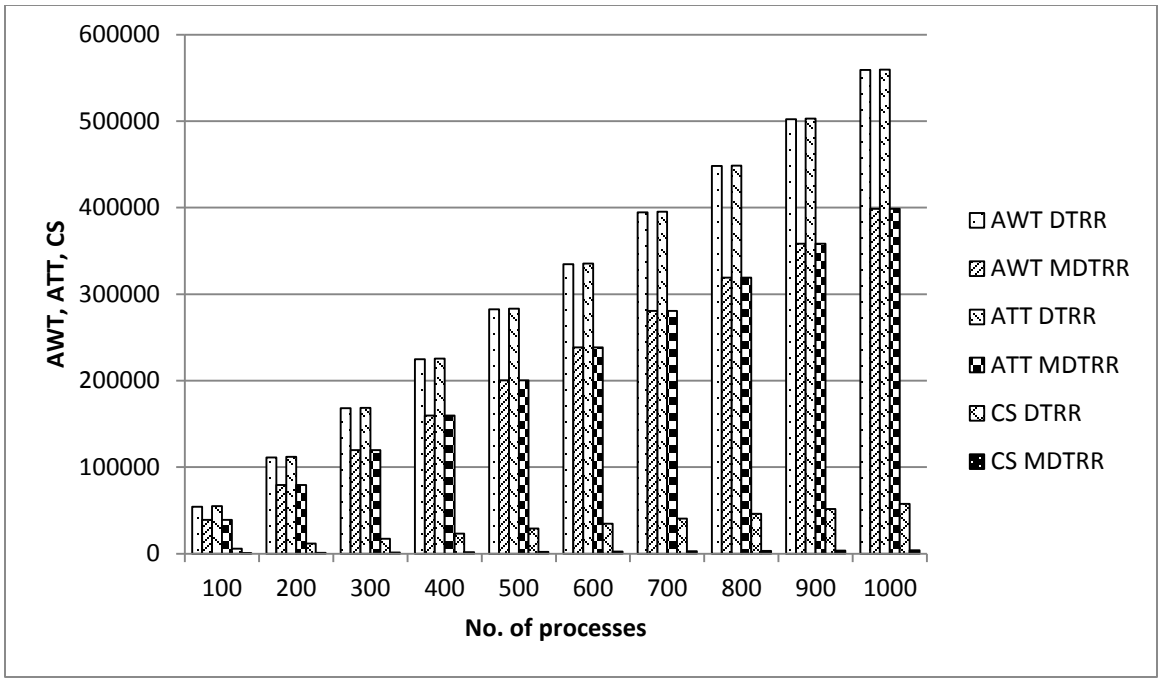

Figure 5.8: Graph for Table 5.8

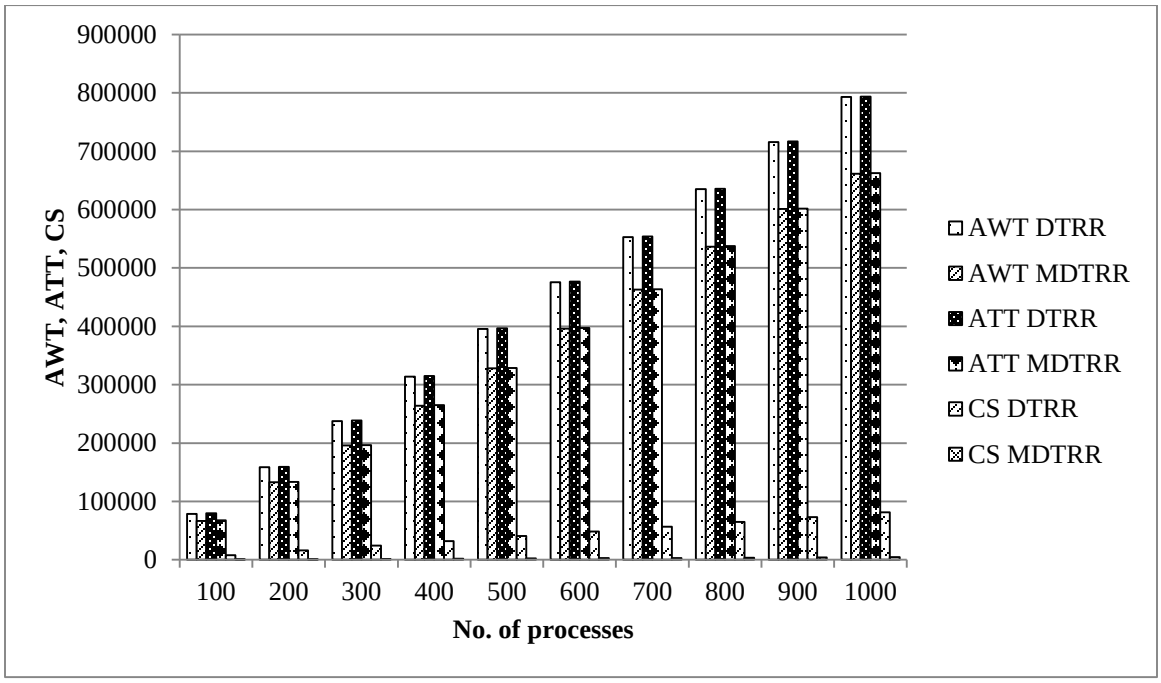

Figure 5.9: Graph for Table 5.9

If we look at case 3; in figure 5.7, the average waiting time of MDTRR is 8.83% to 11.91% lower and average turnaround time is 8.80% to 11.75% lower with respect to DTRR; in figure 5.8, the average waiting time of MDTRR is 39.58% to 40.96% lower and average turnaround time is 38.99% to 40.83% lower with respect to DTRR and in figure 5.9, , the average waiting time of MDTRR is 17.72% to 20.62% lower and average turnaround time is 17.50% to 20.56% lower with respect to DTRR. In all three figures, the context switch decreases drastically. Here also, in figure 5.8, AWT and ATT are decreased by greater percentage than in figure 5.7 and 5.9, because in some condition modified algorithm $q=q^* \lambda /f$ increases the value instead of decreasing for better result.

More importantly, in all three cases, the percentage value increases or decreases mainly due to the selection process of Q.

CHAPTER SIX

CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

Several attempts have been made to improve the round robin scheduling with fixed quantum. In basic version of round robin scheduling algorithm, if the size of time quanta is decreased, the number of context switches is increased. The relation between number of context switch and size of time quanta is inversely proportional to each other. Now when the time quanta for RR is calculated dynamically, then the calculation of dynamic quanta can be done with various measures such as mean, median, mode, quartile, percentage requirement, etc. DTRR uses the time Q for one complete round of all active processes is to keep constant. From Q and number of jobs 'n', we calculate quantum q as q= Q/n. Here the relation between q and n is linear. i.e. quantum q as inverse relation with the number of jobs in ready queue. This dissertation work modifies the dynamic quantum selection strategy of DTRR policy as below:

$$\text{Quantum}(q) = \begin{cases} q * \frac{\lambda}{f} & \text{if } f \geq 0.0625 \\ 4q & \text{if } f \leq 0.0625 \end{cases}$$

This equation is mainly focused on fraction of jobs finished in previous round of RR scheduling algorithm. From this it is clear that quantum value used in MDTRR is optimum than the quantum value used in DTRR.

The performance of DTRR is compared with MDTRR; it is found that average waiting time of MDTRR is 3.52% to 40.96% less than DTRR scheduler and average turnaround time is 4.89% to 40.83% less than DTRR scheduler. Besides this, MDTRR uses optimum quantum size than DTRR policy and at the same time number of context switches decreases drastically in MDTRR policy. This is another great achievement of this dissertation work.

Hence the dissertation is successful to experimentally verify that MDTRR policy is far superior than DTRR policy.

6.2 Recommendation

Now the further work may be to investigate the other statistics to calculate the size of dynamic quanta for round robin scheduling such as mean, mode, quartile, etc and possible modification on these statistics to produce better size of time quanta in which the scheduling algorithm performs better with respect to response time, waiting time, turnaround time and number of context switch. Another interesting further work is to make quantum dynamic on the basis of λ and f value. Idea is to increase the quantum size if there is some other value of λ and f.

References:

[1] Behera, H. S., Mohanty, R., Bhoi, S. K., Sahu, S., "Comparative performance analysis of Multi- Dynamic Time Quantum Round Robin algorithm with arrival time", Indian Journal of Science and Engineering, 2(2), 262-271, 2011

[2] Behera, H. S., Mohanty, R., Nayak, D., "A New Proposed Dynamic Quantum with Re-Adjusted Round Robin scheduling algorithm and its performance analysis", International Journal of computer Applications(0975-8887) volume 5-No.5, August 2010

[3]Behera, H. S., Panda, Jajneseni, Thakur, Dipanwita and SahooSubasini, "A New proposed Two Processor Based CPU Scheduling Algorithm with Varying Time Quantum for Real Time Systems", Journal of global research in computer science(JGRCS), ISSN-2229-371 X, volume2, No.4, April 2011

[4] C. Yaashuwanth, and R. Ramesh, "Intelligent Time Slice for Round Robin in Real Time Operating Systems", IJRRAS, 2(2), 126-131, 2010

[5] Dhal, Shwetasonali, Pallabbenerjee, "Comparative performance analysis of Average Max Round Robin Scheduling algorithm using Static Time Quantum", IJITEE, ISSN-2278-3075, volume-1, Issue-3, August 2012

[6] Ghishing, Ashim,"Analysis of the Varying Time Quantum Round Robin Scheduling", Master degree in Computer Science and Information Technology thesis; CDCSIT, TU

[7] Jason Neigh, Chris Waill, and HuaZhong, "Virtual-time round robin: An O(1) proportional share scheduler", In Proceeding of the 2001 USENIX Annual Technical Conference, June, 2001

[8]Mostafa, S. M., Rida, S. Z. and Hamad, S. H. "Finding time quantum of round robin CPU scheduling in general computing systems using integer programming", International Journal of Research and Review in Applied Science. 5(1), 64-71, 2010

[9] Nayak, D., Malla, S.K. and Debadarshini, D., "Improved Round Robin Scheduling using Dynamic Time Quantum", International Journal of Computer Applications, Volume-38, No.5, 2012

[10] Noon, A., Kalakech, A. And Kadry, S., "A New Round Robin Based Scheduling algorithm for operating systems: Dynamic Quantum time Mean Average", International Journal of Computer Science Issues. 8(3), 224-229, 2011. Issues-8(3), 224-229, 2011

[11] Panda, Sanjaya Kumar, Bhoi, Sourav Kumar, "An Effective Round Robin Algorithm using Min Max Dispersion Measure", International Journal of Advanced Research in computer science and software engineering, Volume-2, Issue-11, November, 2012

[12] Rami, J., Matarneh, "Self-Adjusted Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes", Department of Management Information Systems, American Journal of Applied Sciences 6(10): 1831-1837, ISSN-1546-9239, 2009

[13] ShahramSaeidi, HakimehAlemiBaktash, "Determining the Optimum Time Quantum Value in Round Robin Process Scheduling Method ", Information Technology of Computer Science, volume-10, 67-73, 2012

[14] Silberschatz, A., Galvin, P. B., and Gagne, G., Operating Systems Concepts, 7th Edition, John Wiley and Sons, USA

[15] Stalling, W., Operating Systems Internals and Design Principles, 5th Edition

[16] Tanenbaun, A. S., Modern operating Systems, 3rd Edition, Prentice Hall, ISBN: 13: 9780136006633, PP:1104, 2008

[17]Varma, P. Surendra, "A Best Possible Time Quantum for Improving Shortest Remaining Burst Round Robin(SRBRR) Algorithm", International Journal of Computer Science and Engineering(IJCSE).