CHAPTER I

Introduction

1.1Background

Real-Time systems are those systems where not only the correct execution of tasks matters but also the timely execution of the tasks should be accurate. Real-Time computation must satisfy time constraints. Violating the time constraints is as bad as wrong computation [27]. This timely execution of any tasks is determined by deadline of the tasks that is known in prior. Real-Time systems differ from general computer systems in the sense that they must react to events originating in the physical world within certain duration of time. A typical Real-Time system monitors and controls some external process, and the system must notice and respond to changes in the external process in a timely manner, usually on the order of tens or hundreds of milliseconds but sometimes on the order of seconds or on the order of milliseconds [3]. If the external process is simple, then a single micro-computer which responds quickly to a few types of external events might suffice. However, many Real-Time systems are more complex and require more processors, more sophisticated software structure, and a more sophisticated means of coordination among multiple activities; this motivates the need for methods of scheduling various tasks in Real-Time systems [18]. The scheduling is complicated by the fact that some Real-Time systems are such that a breakdown of the system scheduler (due to overload or poor design) can cause a catastrophic failure in the physical system, resulting in loss of life and property.

Real-Time systems are broadly characterized into two groups:

Hard Real-Time Systems:

A Hard Real-Time system is a Real-Time system that must meet its deadlines with a near-zero degree of flexibility. The deadlines must be met, or catastrophes occur. The cost of such catastrophe is extremely high. The computation results obtained after the deadline have either a zero-level of usefulness or have a high rate of depreciation as time

moves further from the missed deadline before the system produces a response. Example of such systems may be nuclear plants, fly-by-wire vehicles etc

Soft Real-Time Systems:

A Soft Real-Time system is a Real-Time system that must meet its deadlines but with a degree of flexibility. The deadlines can contain varying levels of tolerance, average timing deadlines, and even statistical distribution of response times with different degrees of acceptability. In a Soft Real-Time system, a missed deadline does not result in system failure, but costs can rise in proportion to the delay, depending on the application. Example of such systems may be ticket reservation, banking systems etc.

1.1.1Characteristics of Real-Time systems

a) Embedded Systems:

These systems work as an integrated part of some other larger systems. Those larger systems may be electrical, mechanical, electromechanical or any other type. The embedded systems on those larger systems work as a supportive systems for them. Most embedded systems consist of small micro-controller and limited software situated within some product such as microwave oven or automobile.

b) Control Systems:

These systems must react to the environment in time. These systems are used to protect from damages or harms to any system e.g. Air control systems make sure the working of air traffic without a mishap.

c) Reactive Systems:

These systems continuously interact with the environment. Manufacturing industries are the example of these systems. The main characteristic of these systems is the timeliness of the system processing.

d) Safety-critical Systems:

In these systems, the failure may cause damage of system or life. Their must be some mechanism for the recovery of the faults, whenever they occur. In Safety-Critical applications, where the safety of the system-at-large (*e.g., an airplane or a car*) depends

on the correct operation of the computer system (*e.g., the primary flight control system or the by-wire-system in a car*)

1.1.2 Real-Time Operating Systems (RTOS)

A Real-Time Operating System (RTOS) is a program that schedules execution in a timely manner, manages system resources, and provides a consistent foundation for developing application code. RTOS is the top level software system that carries out the scheduling. Application code designed on an RTOS can be quite varied, ranging from a simple application for a digital stopwatch to a much more complex application for aircraft navigation. Good RTOSes, therefore, are scalable in order to meet different sets of requirements for different applications.

The Real–Time Operating System must provide basic support for guaranteeing real-time constraints , supporting fault-tolerance and distribution, and integrating time-constrained resource allocations and scheduling across a spectrum of resource types, including sensor processing, communications, CPU, memory, and other forms of I/O. The key feature of the RTOS is scheduling the tasks in time constraints.

The prime difference between General-Computing operating systems and Real-Time Operating Systems is the need for “deterministic” timing behavior in the Real-Time systems. Formally, “deterministic” timing means that operating system services consume only known and expected amounts of time. General-computing non RTOSes are often quite non-deterministic. Many RTOS supporters argue that a RTOS must not use virtual memory concepts, because paging mechanics prevent a deterministic response. While this is a frequently supported argument, it should be noted that the term "Real-Time Operating System" and determinism in this context covers a very wide meaning, and vendors of many different operating systems apply these terms with varied meaning [7, 9, 19]. Real-Time Operating Systems are often uses in embedded solutions, that is, computing platforms that are within another device. Examples for embedded systems include combustion engine controllers or washing machine controllers and many others. Desktop PC and other general-purpose computers are not embedded systems. While Real-Time Operating Systems are typically designed for and used with embedded

systems, the two aspects are essentially distinct, and have different requirements. A Real-Time Operating System for embedded system addresses both sets of requirements. Study of the Real-Time Operating System is equally important as the study of real-time scheduling theory.

Real-time Tasks

A task is an independent thread of execution that can compete with other concurrent tasks for processor execution time. Real-time tasks are defined as logical executable unit of any program running on processor of Real-Time systems. Hard Real-Time tasks are those tasks that are running on Hard Real-Time systems where the tasks should be completed on given hard deadline. Soft Real-Time tasks are those tasks that are running on soft Real-Time systems where the incompletion of task on given timing constraint will not cause any fatal error, though the incompletion on given deadline is not desired.

Life Cycle of a task

When a task is loaded in memory, it becomes ready to execute. When the scheduler selects the task for execution, the task enters the running state. In this state, the task can either preempt which is the case when its priority is higher at next period. When a task is preempted then the operating system puts the task on the end of the ready queue of tasks, but it remains ready to execute.

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

Ready:

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

Running:

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

Blocked:

This state is a consequence of the lowest priority of the task. The execution of task is halted. Tasks in this state are waiting to be queued-up in ready state.

Types of Real-Time tasks

Following are the types of tasks on the basis of periods of the tasks.

a) Periodic tasks:

Periodic tasks are real-time tasks which are activated or released regularly at fixed rates or periods. Normally, periodic tasks have constraints which indicate that instances of them must execute once per period P.

b) Aperiodic tasks:

Aperiodic tasks are real-time tasks which are activated irregularly at some unknown and possibly unbounded rate. The time constraint is usually deadline D.

c) Sporadic tasks:

Sporadic tasks [6] are real-time tasks which are activated irregularly with some known bounded rate. The bounded rate is characterized by a minimum inter-arrival period, that is, a minimum interval of time between two successive activations. The time constraints are usually a deadline D.

Following are the types of tasks on the basis of interruptions.

a) Preemptive tasks:

A task is preemptive if it interrupts the currently executing task and it starts executing because of its higher priority than currently executing task. Preemption is generally done by hardware interruption generated by RTOS.

b) Non-preemptive tasks:

A task is non-preemptive if it waits for currently executing tasks to finish its execution despite its higher priority. Currently executing task had started executing before than the requesting task

1.2 RTOS Scheduler

The scheduler is at the heart of every kernel. A scheduler tracks the scheduling algorithms needed to determine which task executes when. To understand how scheduling works, this section describes the following topics.

1.2.1 Schedulable tasks

A schedulable entity is a kernel object that can compete for execution time on a system, based on a predefined scheduling algorithm. Tasks and processes are examples of schedulable entities found in most kernels.

1.2.2 Multitasking

Multitasking is the ability of the operating system to handle multiple activities within set deadlines. A real-time kernel might have multiple tasks that it has to schedule to run.

1.2.3 Context switching

Each task has its own context, which is the state of the CPU registers required each time it is scheduled to run. A context switch occurs when the scheduler switches from one task to another or when the lower priority task is preempted by the higher priority task.

1.2.4 Dispatcher

The dispatcher is the part of the scheduler that performs context switching and changes the flow of execution. At any time an RTOS is running, the flow of execution, also known as flow of control, is passing through one of three areas: through an application task, or through the kernel [7].

1.3 Overview of Algorithmic Real-Time scheduling

Scheduling involves the allocation of resources and time to tasks in such a way that certain performance requirements are met. Scheduling determines order of execution of the tasks. The basic problem in Real-Time systems is to make sure that tasks meet their

time constraints. Real-Time Scheduling, thus, needs to schedule the tasks keeping in mind about the time constraints of the tasks. More information about the tasks is known such as:

- Number of tasks
- Resource Requirements
- Release Time
- Execution time
- Deadlines.

The scheduling techniques in Real-Time Systems are categorized as follows, according to the nature of the machine

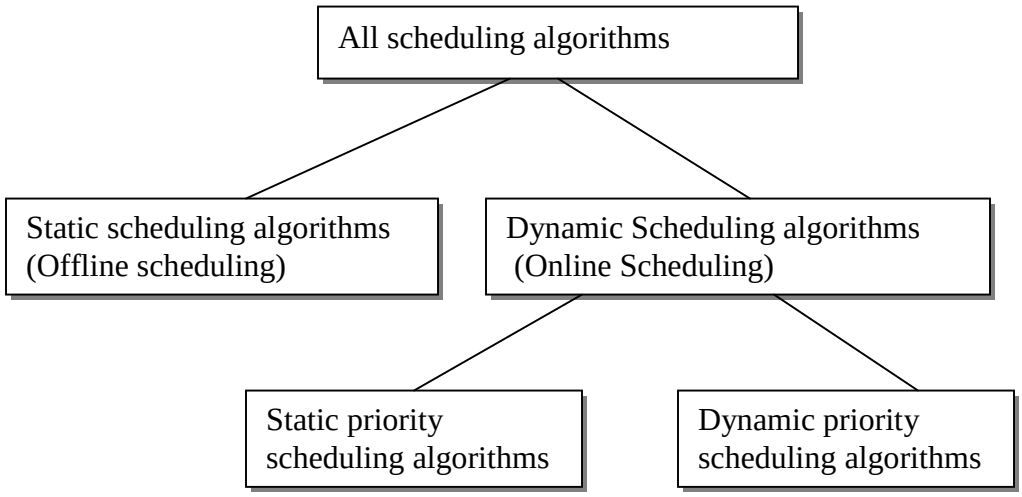

Figure 1.1:- *Classification of Scheduling Algorithms*

1.3.1 Uniprocessor scheduling

Uniprocessors are those systems where exactly one processor is available. All the tasks in the systems are required to execute in this single shared processor. The set of uniprocessor real-time scheduling algorithms is divided into two major subsets, namely off-line scheduling and on-line scheduling.

1.3.1.1 Off-line scheduling (Pre-run-time scheduling):

These generate scheduling information prior to system execution. The scheduling information is then utilized by the system during runtime [22]. In this type of scheduling, feasible schedule is computed off-line based on a priori knowledge of the tasks' characteristics.

In systems using off-line scheduling, the ordering of the execution of tasks is, generally, required. This can be accommodated by using precedence relationships that are enforced during off-line scheduling.

1.3.1.2 On-line scheduling

These algorithms generate scheduling information while the system is running. The on-line schedulers do not assume any knowledge of tasks' characteristics until they arrive to the processor. These algorithms require large amount of run-time processing time [22].

On-line scheduling algorithms can be divided into Static-priority based algorithms and Dynamic priority based algorithms, which are discussed as follows,

a) Static-priority based algorithms:

A scheduling algorithm is said to be static if the priorities of the individual tasks are assigned to tasks, usually at system startup and initialization. These priorities could be assigned in different ways. For example, it could be based on the worst case execution time of tasks, period of tasks, or how critical it is for a task to execute. The priority assignments of the Rate Monotonic Algorithm are based on the period of the tasks [16]. In this case, the task with the shortest period has the highest priority.

b) **Dynamic-priority based algorithms**:

A scheduling algorithm is said to be dynamic if the priorities of the individual tasks might change from request to request [16]. Again, the way in which these priorities are determined may vary, and could be based on task deadlines or whatever criteria utilized by the scheduling algorithm. Earliest Deadline First is widely used Dynamic-priority based algorithm used in Real – Time systems.

1.3.2 Multiprocessor Scheduling Algorithms

In multi-processor platforms there are several processors available upon which the tasks may execute [22]. Scheduling Multiprocessor systems requires executing all its tasks in shared processor or shared memory and sharing I/O environment which is considerably quite different from uniprocessor systems.

1.4 Scheduling goals

In terms of schedulers, there is no single definition of performance that fits everyone's needs i.e., there is not a single performance goal for the scheduler to achieve. The many definitions of good scheduling performance often lead to a give-and-take situation; for instance, improving performance in one way decreases performance in another. Here, the dissertation is trying to achieve the knowledge of performance in three different ways, namely, processor utilization, turnaround time and waiting time.

Following goals should be considered in scheduling Real-Time systems:

- Meeting the timing constraints of the system
- Preventing simultaneous access to shared resources and devices attaining a high degree of utilization while satisfying the timing constraints of the system.
- Reducing the cost of context switches caused by preemption.

1.5 Motivation and Aims

The world is growing with Real-Time applications and for Real-Time systems, the *real time* of the systems is determined by the scheduling of the tasks executing in the system. So, analytical study of the scheduling algorithms would be helpful for deciding their appropriate use for the Real-Time applications.

The optimality and the feasibility of the Real-Time scheduling algorithms are determined by the minimum turnaround time, minimum waiting time, minimum context switches and maximum processor utilization. Different task-sets with different parameters are to be subjected to the Evaluation Model (simulator) so that the performance analysis to

conclude the optimality and the feasibility of the Real-Time Scheduling algorithms will be carried out.

To obtain the optimality and the feasibility of the Real-Time scheduling algorithms, the dissertation simulates the characteristics of Rate Monotonic scheduling algorithm and Earliest Deadline First scheduling algorithm. The dissertation figures out the performance of Rate Monotonic Scheduling over Earliest Deadline First scheduling and other priority-based scheduling algorithms. The step-by-step execution of each scheduling algorithm for particular task set makes simple for evaluation of the performance of the scheduling algorithms.

1.6 Thesis Organization

The rest of the material of the study is organized into subsequent seven chapters. Chapter 2 describes what study has been made to get the dissertation done. The study of current theories and materials is shown in this chapter. Chapter 3 incorporates the description of evaluation model. The key job, in this dissertation, is the design of a simulation model that helps to analyze the two real-time scheduling algorithms, Rate Monotonic Scheduling and Earliest Deadline First Scheduling with other general scheduling algorithms. This chapter specifies the composition of the evaluation model. The design of the evaluation model upshots the simulator for the analysis of Real-Time scheduling algorithms. Chapter 4 describes the design of the evaluation model specified in chapter 3. This chapter explains the components formulated for the design of the evaluation model. Chapter 5 integrates the implementation details of the development of simulator. The description about tools that are used for constructing the simulator and the development model is detailed. In Chapter 6, the analysis of the algorithms is described. In other words, the working of the simulator is observed. The results of the outcome of the algorithms with respect to different tasks sets is observed and described in this chapter. This chapter covers the core part of the dissertation. Chapter 7 concludes the results of the dissertation and makes further recommendations. Chapter 8 has references of books and papers studied and used to accomplish the dissertation.

CHAPTER II

Methodology

The foundation of this dissertation is priority based real-time scheduling algorithms, because of their sound mathematical basis and superiority over other scheduling schemes in terms of predictability and flexibility. This chapter describes the study of the existing materials regarding the concepts of the Real-Time systems and scheduling theory that assisted to complete the dissertation. The dissertation is based on the experimental analysis of the real-time scheduling algorithms.

Section 2.1 describes the study about schedulability analysis, implementation complexity, runtime overhead and Real-Time Operating Systems that has been made. The study of these terminologies paved the way for the progress of the dissertation. Section 2.2 describes the statistics that are used in the dissertation, as the analytical experimentation part, which are the key elements for the evaluation of the scheduling algorithms. The scheduling algorithms are implemented concerning with these statistics and the effect is analyzed later on. Algorithm evaluation method is described in Section 2.3 which gives the idea of the method the dissertation has tracked to evaluate the algorithms. Next section 2.4 describes the predictability property of the system. Predictability verifies the timing behavior of the systems to be assured in the system. Section 2.5 has the description of the works that are related to the works. The idea in these works helped to enhance the preparation of this dissertation work. Finally Section 2.6 illustrates the simulator that is constructed for the analysis purpose. This is a java applet program that works as a scheduler.

The methodology adopted during this dissertation is mainly:

- Analytical reasoning and theoretical derivation of schedulability tests.
- Explanation of results using examples.
- Evaluation using simulation.

2.1 Literature Survey

In this section, the basic formulation of the scheduling theory is described. The more study about the classification of the scheduling theory is explained in [22]. Since the scheduling is considered in Real-Time systems, only scheduling techniques that approach to the systems are considered in this dissertation.

2.1.1 Scheduling Problem

The evolution of scheduling closely tracked the development of computers. The concept of scheduling is not new; Henry L. Gantt (see [11]), an American engineer and social scientist is credited with the development of the bar chart (Gantt chart) in 1917 to show the performance of different scheduling algorithms.

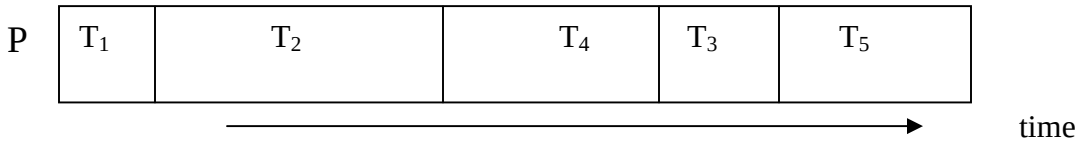

Figure 2.1: *Gantt chart representation for a schedule of five tasks*

In above figure, five tasks are scheduled in Gantt chart representation. The tasks are ordered in occurrence of time of execution.

This dissertation studies on the scheduling problem regarding the Real-Time systems. The scheduling techniques based on other Operating Systems are not studied that much, for this dissertation.

Priority based scheduling algorithms are one of the oldest, simplest, and most widely used algorithms in Real-Time systems. There are many variations of the priority based scheduling algorithms. For example, Rate Monotonic Scheduling algorithms (RM), Earliest Deadline First (EDF) scheduling algorithms are the widely used Real-Time scheduling algorithms.

2.1.2 Schedulability Analysis

The schedulability determines whether a set of tasks meets its timing constraints. Liu and Layland [2] showed that RMS is optimal in the sense that if the RMS priority assignment is not feasible, a set of tasks is not schedulable. They also discovered the least

upper bound on processor use in a static priority scheme. More precisely, if, for a set of n tasks, the processor use is less than $n(2^{1/n}-1)$, that set of tasks is schedulable.

$$\sum_{i=1}^{N} U_i \le n\,(2^{1/n} - 1) \qquad \text{where, } U_i = \sum_{i=1}^{n} C_i/T_i$$

and U_i is a utilization of i tasks.

Liu and Layland have also found an even stronger utilization result for a dynamic priority assignment policy called earliest deadline first (EDF). The basic assumption of EDF is that at any point in time the priority of an enabled task is not fixed; it depends on the time until the next request for that task. According to the EDF policy, the executing task must always have the least time remaining until the next request or equivalently the earliest deadline) among all the enabled tasks. An EDF-executed set of tasks is schedulable if, and only if, its processor use is less than 1.

i.e.

$$U_i = \sum_{i=1}^{n} C_i/T_i \le 1$$

This schedulability test is based on the optimality of the EDF. The optimality of EDF denotes that there are no deadline misses for any task [2, 3, 4].

2.1.3 Implementation complexity

In RMS, the ready queue is ordered decreasing fixed priority levels, under EDF it has to be ordered increasing *absolute deadlines*. Thus, once the *absolute deadline* is available in the task control back, the basic kernel operations (e.g. insertion, extraction, dispatch) have the same complexity, both under RMS and EDF.

An advantage of RMS over EDF is that, if the number of priority levels is not high, the RMS algorithm can be implemented more efficiently by splitting the ready queue into several FIFO queues, one for each priority level. In this case, the insertion of the task in the ready queue can be performed in O (1). Unfortunately the same solution cannot be

adopted for EDF, because the number of queues would be too large (e.g. equal to 2^{32} if system time is represented by four byte variables).

A disadvantage of EDF is that absolute deadline changes from task to another task and need to be computed at each task activation. Such a runtime overhead is not present under RMS, since periods are typically fixed [4, 5].

2.1.4 Runtime Overhead

Runtime overhead is the extra time or resources (memory, I/O, processor) consumed by the task in the middle of its execution. This extra consumption reduces the efficiency and the performance of the scheduler of the Real-Time Operating System.
In this dissertation, the computation of the runtime overhead is considered by context switches, the waiting time jitter and the effectiveness in improving the periodic responsiveness of the tasks. The overhead due to the context switches attracts the attention to resolve the conclusion.

2.2 Statistics

These are the numeric entities the *task set* holds for each of its tasks. The whole evaluation of the dissertation is based on the statistics presented below. These statistics are the evaluation criteria the evaluation is done with respect to. These are the key parameters around which the analytical study is done in this dissertation.

2.2.1 CPU utilization

It is the percentage of time for which CPU is busy with tasks. Here, we want to keep the CPU as busy as possible. Thus, if the running task requests for I/O operation, then, another task is selected to execute so that CPU is kept busy. Concept of multiprogramming is used for maximizing the CPU utilization.

Several tasks are kept in memory and are thus ready to run. Scheduling time is, of course, an overhead since no useful work is done. Utilization is thus measured by throughput which is measured as the number of tasks completed per unit time.

2.2.2 Turnaround Time

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

2.2.3 Finish Time

It is the time by which task should be completed. A task that is waiting and ready is dropped if its completion deadline is passed.

2.2.4 Waiting Time

Waiting time is total time the task has consumed to finish its execution once it is subjected to the processor. The waiting time of a task in a particular schedule is defined to be the amount of time that has elapsed between the arrival of the task and its completion. Clearly, in order for a schedule to meet all deadlines it is necessary that the waiting time of each task not exceed the relative deadline of the task.

2.2.5 Context Switches

Context of the task consists of program counter, stack pointer as well as program status word and processor register. It can also include memory block addresses. More context switches means more saving the interrupted task context and retrieving or loading the new context of the called task.

Context switch time is defined as the amount of time it takes a RTOS to save the context of one task (save its pointers and registers to a control block) and load the context of another task from a second control block. Context switch time never shows up in isolation in a system, it is always overhead that results from a RTOS call.

2.2.6 Completion Deadline

It is the time before which task should have completed its computation. In RMS and EDF the deadline is determined by the period of the task. The violation of the deadline will theoretically wrong computation of the Real-Time systems.

2.3 Algorithm evaluation method

The algorithms are evaluated with respect to criteria, namely, CPU utilization, turnaround time, and waiting time. The evaluation requires a method of taking input data for the algorithm and giving the output with respect to the data.

The model, thus, is a deterministic. Deterministic modeling is one type of analytic evaluation method. This method takes a particular predetermined workload (task set) and defines the performance for that workload (task set) with scheduling algorithms.

2.4 Predictability (guaranteeing the timing behavior)

[7, 19] describe about the predictability. Timing behavior is major issue of Real-Time Systems. Analysis of conditions of failure in satisfying timing constraints points out the predictability of the systems. In Real-Time Systems scheduling, the proper analysis of tasks meeting their deadlines needs much attention. A RTOS is predictable if the time necessary to acknowledge a request of an external event is know in advance. Predictability means that when a task is activated it should be possible to determine its completion time with certainty. It is also desirable that the system attain a high degree of utilization while satisfying the timing constraints of the system.

2.5 Related Works

A schedulability test for a scheduling policy is sustainable if any system deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual tasks are changed in some, or all of the following ways: (i) *decreased execution requirements*; (ii) *later arrival times*; (iii) *smaller jitter*; and (iv) *larger relative deadlines* [32].

Feasibility analysis of fixed priority systems has been widely studied in the real-time literature and several acceptance tests have been proposed to guarantee a set of periodic tasks. They can be divided in two main classes: polynomial time tests and exact tests. Polynomial time tests are used for on-line guarantee of dynamic systems, where tasks can be activated at runtime. These tests introduce an adequate overhead, when executed upon a new task arrival, however, provide only a sufficient schedulability condition, which may cause poor processor utilization. On the other hand, exact tests, which are based on waiting time analysis, provide a necessary and sufficient schedulability condition, but are too complex to be executed online for large task sets. As a consequence, for large task sets, they are often executed offline [28].

Scheduling about aperiodic tasks and sporadic tasks resemble the periodic task scheduling though their rules vary according to the nature of the tasks. These tasks are defined in chapter 1. Their scheduling also comprises the part of the Real-Time systems scheduling. These scheduling techniques are studied in [3].

Deadline Monotonic Algorithm assigns to tasks priorities that are inversely proportional to their deadlines. The schedulability test is more complicated that that suggested by Liu and Layland and involves testing for all possible task phasings [41].

Priority Inversion can result from mutual execution when high priority tasks have to wait for the lower priority independent tasks. Priority inversion on such critical section can be handled by *priority inheritance protocol*.

2.6 Simulator (Evaluation Model)

The simulator is an *Evaluation Model* that works as a scheduler for Rate Monotonic Scheduling, Earliest Deadline First and other scheduling algorithms. The simulator is an applet that demonstrates the animation of scheduling to the timeline. Despite this, the simulator also gives the statistical results after computation of the required parameters. The goal of the simulator is to make it possible to evaluate the performance of priority-based scheduling algorithm by observing the changes in the selected parameters with different scheduling techniques.

CHAPTER III

Specification

In the dissertation, the construction of the Evaluation Model is done as part of the empirical analysis. The construction of the Evaluation Model is an integration form of the implementation of the scheduling algorithms. This Evaluation Model is not machine oriented, means, the simulator is not designed following the core ideas of kernel. The Evaluation Model does not directly deal with the basic functionalities of the operating system such as input/output, interrupt handling, scheduling, main and auxiliary storage management, process and resource data structure. It is just a simulator which shows the functioning of the scheduling algorithms in applet, a java program.

In the following sections, brief specification of Evaluation Model is featured. In section 3.1, Evaluation Model is characterized with the machine specification, algorithms it runs and its working nature. It has some limited working criteria as assumed in the section 3.2. Section 3.3 explains the scheduling algorithms besides RMS and EDF. These scheduling algorithms play a supportive role in analytical evaluation for this dissertation.

3.1 Evaluation Model characterization

The Evaluation Model that is developed works for Uniprocessor Real-Time Systems. The Evaluation Model is a multi-scheduler simulator that is the consequence of implementations of the priority based scheduling algorithms and generates the statistics to illustrate the performance analysis. Evaluation Model is an applet, designed in Java Programming language as a simulator for Rate Monotonic Scheduler, Earliest Deadline First scheduler and other priority based scheduling algorithms like Round Robin, First Come First Serve, Shortest Remaining Time First, and Shortest Process Next. The applet generates the statistics of the scheduling algorithms for a particular task set. The task set is a predefined set of tasks, with respective information about the tasks, which is used for the implementation of a scheduler.

3.2 Assumptions

The Evaluation Model has a limited exposure to the working of the scheduling techniques. Though, it tries to broaden the coverage of adequate scheduling features, some of them could not be covered. As a result, the Evaluation Model handles the scheduling of tasks only if the task set has the tasks with following conditions.

1. Each periodic task completes within its period.
2. No task is dependent on any other task.
3. Each task needs the same amount of processor time on each execution burst.
4. Any non-periodic tasks have no deadlines.
5. Task preemption occurs instantaneously and with no overhead.
6. No consideration to memory or I/O has been made.
7. At least one of the tasks in task-set starts with 0 arrival time.

3.3 Other scheduling algorithms

These are the priority based common scheduling algorithms used by general operating systems. Their possibilities of use in the Real-Time systems are heavily studied in the dissertation. These scheduling algorithms are also analyzed in terms of processor utilization, turnaround time and waiting time. Following are the scheduling algorithms that are used for analytical purpose in this dissertation.

a) Shortest Task Next (STN):

This scheduling algorithm assigns its highest priority to the task which has the smallest execution time to execute next. This is a non-preemptive algorithm in which the process with the shortest expected executed time is selected next. Thus a short task will jump to the head of the queue past longer jobs.

b) First Come First Serve (FCFS):

With this algorithm, processes are assigned the CPU in the order they request it. Basically, there is a single queue of ready processes. When the first task enters to the scheduler, it is started immediately and allowed to run as long as it wants to. As other

tasks come in, they are put onto the end of the queue. When the running task blocks, the first task on the queue is run next. When a blocked task becomes ready, like a newly arrived job, it is put on the end of the queue.

c) Shortest Remaining Time (SRT):

A preemptive version of shortest task next is shortest remaining time next. With this algorithm, the scheduler always chooses the task whose remaining run-time is the shortest. Again here, the run time has to be known in advance. When a new task arrives, its total time is compared to the current tasks' remaining time. If the new task needs less time to finish than the current process, the current process is suspended and the new task started. This scheme allows new short tasks to get good service.

d) Round Robin (RR):

Each task is assigned a time interval, called its quantum, which it is allowed to run. If the task is still running at the end of the quantum, the CPU is preempted and given to another task. If the process has blocked or finished before the quantum has elapsed, the CPU switching is done when the task blocks, of course. A clock interrupt is generated at periodic intervals. When the interrupt occurs, the currently running task is placed in the ready queue, and the next ready task is selected on an FCFS basis. In the dissertation RR is implemented with time intervals of 1 and 4.

CHAPTER IV

Design of Evaluation Model

This chapter describes the features that are used as part of the design of the Evaluation Model based on the specification portrayed in chapter 3. Design of Evaluation Model has the motive to act in accordance with the need of the construction of a simulator that outputs the statistical results for scheduling techniques. The Evaluation Model is designed by considering the modules that construct the model. Since the Evaluation Model is a java applet, the modules are designed using the *Java* features. Basic Java features like *Thread, Graphics etc.*

Section 4.1 describes the modules that are modeled for the construction of the Evaluation Model. The section clarifies the phenomenon of the all seven modules that are explained individually. The algorithms of the modules given on the section are supposed to provide the comprehensive way of understanding the process of each module. These modules compose the formation of the simulator. These modules act as parts of the simulator. Section 4.2 describes the integration model of these modules. As described earlier, all these modules are integrated to accomplish the purpose of the Evaluation Model. Section 4.3 states that the Evaluation Model is a deterministic model.

4.1 Modules of Evaluation Model

Modules that are defined in this section have a meaning of logical entities that design the simulator. Each module has its own functionality, though, sometimes, the concept may seem only logical. Different modules have the different functionality like taking input for the scheduling algorithm, making computation, giving output, painting the scheduling timeline execution.

Each module is coded separately and integrated as a package later on. Following are the modules that construct the simulator.

4.1.1 graphCanvas

This module carries out the painting function to draw the timeline execution of the tasks. It uses the *abstract window tool* (AWT) provided by the Java Programming Language [35]. It encapsulates a blank window which it can be drawn upon. It uses the *Graphics* class to inherit the graphics context, which demonstrates the graphics environment the applet is running in.

A function *paintBoard( )* is used for drawing the timeline of the task set for the particular scheduling algorithm. It takes the parameter as object *Graphics* and draws the timeline in the applet.

The coding detail of the module is summarized in Appendix A.

Algorithm

Step 1 import attributes and methods from Canvas and Graphics

Step 2 if (scheduling algorithm)

        import attributes for paint() method from Graphics

        draw the execution of tasks; paintBoard() function

Step 3 end

Listing 4.1:- *Algorithm for graphCanvas module*

4.1.2 GUI (Graphical User Interface)

This module performs the part of User Interface. It uses the *Observable* class to create the subclasses that other parts of the program can observe. When an object of such a subclass undergoes a change, observing classes are notified the effect. This is done by *notifyObservers( )* method.

The coding detail of the module is summarized in Appendix B.

Algorithm

Step 1. Inherit attributes from *Observable* object

Step 2. GUI constructor

return the status of the calling object to “unchanged”;

clearChanged() method

Step 3. method input(object)

call when the calling object has changed; setChanged() method;

notifyObservers method(object);

Step 4. end

Listing 4.2:- *Algorithm for GUI module*

4.1.3 Packet

This module formats the task set. This module is responsible for taking the inputs of the scheduling, setting up them and packet*ing* them for further manipulation. The input parameters like task name, arrival time, deadline, service time, algorithm name are set by this module.

The coding detail of the module is summarized in Appendix C.

Algorithm

Step 1. Constructor Packet

Initialize task ← task name

Initialize arriv ← arrival time of task

Initialize serv ← service time of task

Initialize alg ← algorithm name of task

Initialize sdead ← deadline of task of task

Step 2. function getalgorithm name()

return alg;

Step 3. function gettaskname()

return task;

Step 4. function getarrivaltime()

return arriv;

Step 5. function getservice time()

return serv

Step 6. function getalgorithm()

return alg;

Step 7. function gettaskdeadline()

return sdead;

Step 8. end

Listing 4.3:- *Algorithm for Packet module*

4.1.4 Algorithm configuration

This module is a central part of the Evaluation Model. It is a core implementation of the scheduling algorithms which will actually *perform* the scheduling. This module integrates all the instantiations of other modules to form the organization of the Evaluation Model. Apart from the implementation of the scheduling algorithms, this module is responsible for running the simulator. The working of the simulator is signified by this module.

The coding detail of the module is summarized in Appendix D.

Following scheduling algorithms are implemented as part of the analysis in this dissertation. The detail of the algorithms is presented in previous chapters.

a) Rate Monotonic Algorithm

Algorithm:

Step 1. Input a task set

Initialize the packet class with input parameters

Step 2. Assign priorities to each task

Step 3. Queue up the tasks in ready state with arrival time

Step 4. while(number of tasks in queue)

Dispatch the tasks to scheduler with current arrival time

Task with highest priority in ready state will execute next

Terminate tasks with finished execution

Queue up tasks to ready state for next instance

Step 5. End

Listing 4.4.1:- *Algorithm for Rate Monotonic Scheduling*

b) Earliest Deadline First Algorithm

Algorithm:

Step 1. Input a task set

Initialize the packet class with input parameters

Step 2. Compute deadlines for each tasks

Step 3. Queue up the tasks in ready state with arrival time

Step 4. while(number of tasks in queue)

Compute absolute deadline of tasks for each instance

Dispatch the tasks to scheduler with current arrival time

Task with nearest deadline in ready state will execute next

Terminate tasks with finished execution

Queue up tasks to ready state for next instance

Step 5. End

Listing 4.4.2:- *Algorithm for Earliest Deadline First Scheduling*

c) Round Robin Algorithm

Algorithm:

Step 1. Input a task set

Initialize the packet class with input parameters

Step 2. Assign the quantum (execution limit time) number

Step 3. Queue up the tasks in ready state with arrival time

Step 4. while(number of tasks in queue)

Dispatch the tasks to scheduler with current arrival time

Task with lowest arrival time in ready state will execute next for duration of quantum number

Terminate tasks with finished execution

Queue up tasks to ready state for next instance

Step 5. end

Listing 4.4.3:- *Algorithm for Round Robin*

d) First Come First Serve

Algorithm:

Step 1. Input a task set

Initialize the packet class with input parameters

Step 2. Queue up the tasks in ready state with arrival time

Step 3. while(number of tasks in queue)

Dispatch the tasks to scheduler with current arrival time

Task with lowest arrival time in ready state will execute next

Terminate tasks with finished execution

Queue up tasks to ready state for next instance

Step 4. end

Listing 4.4.4:- *Algorithm for First Come First Serve*

e) Shortest Task Next

Algorithm:

Step 1. Input a task set

Initialize the packet class with input parameters

Step 2. Queue up the tasks in ready state with arrival time

Step 3. while(number of tasks in queue)

Dispatch the tasks to scheduler with current arrival time

Task with lowest finish time in ready state will execute next

Terminate tasks with finished execution

Queue up tasks to ready state for next instance

Step 4. end

Listing 4.4.5:- *Algorithm for Shortest Task Next*

f) Shortest Remaining Time Next

Algorithm:

Step 1. Input a task set

Initialize the packet class with input parameters

Step 2. Queue up the tasks in ready state with arrival time

Step 3. while(number of tasks in queue)

Dispatch the tasks to scheduler with current arrival time

Task with shortest service time in ready state will execute next

Terminate tasks with finished execution

Queue up tasks to ready state for next instance

Step 4. end

Listing 4.4.6:- *Algorithm for Shortest Remaining Time First*

4.1.5 Process

This module computes the output statistics like turnaround time, finish time, waiting time of the task for a particular task set.

The coding detail of the module is summarized in Appendix E.

Algorithm

Step 1. process constructor

Initialize taskNum← Number of tasks in task set;

Initialize name← name of the task;

Initialize ArrivalTime← arrival time of the task;

Initialize ServiceTime← Computed service time;

Initialize Deadline ← deadline of the task;

Step 2. Compute Finish time;

Step 3. Compute Service time;

Step 4. Compute Waiting time;

Step 5. Compute Turnaround time;

Step 6. Compute Servicing time period for each timeline;

Step 7. end

Listing 4.5:- *Algorithm for Process Module*

4.1.6 Scheduler

This module is responsible for making the exact scheduling of the tasks. Manipulation of queues is done to store ready tasks and finished tasks are removed by queue. This also uses the *Vector* class for generating queue. Instantiation of Thread object maintains the scheduling of the tasks.

The coding detail of the module is summarized in Appendix F.

Algorithm

Step 1. Imports the attributes and functions of Vector class

Step 2. Vector class initialization for ready queue and finish queue;

Step 3. Thread implementation for scheduling

Step 4. Scheduler constructor (vector, scheduling_class object, clock)

Step 5. Scheduler constructor (vector, scheduling_class object, clock, algorithm)

Step 6. Compute processready(clock_tick);

Step 7. Resetting of Queue by implementing the GUI object

Step 8. end

Listing 4.6:- *Algorithm for Scheduler Module*

4.1.7 runAlgorithm

This module runs the working of algorithm. The tasks are scheduled according to the algorithm. It uses the *Vector* class to store and manipulate groups of task sets. *Vectors* are the dynamic arrays. This module creates thread by instantiating an object of type *Thread*. Threads are responsible for execution of the tasks complying with the clock time of the processor.

The coding detail of the module is summarized in Appendix G.

Algorithm

Step 1. Imports the attributes and methods of Vector class

Step 2. Inherits the attributes and methods of Scheduler class and implements the Runnable interface

Step 3. RunAlgorithm constructor (vector, scheduling_class object, clock, algorithm)

Step 4. function run()

Queue the name of task currently executing

Set the queue to the timeline

While there is queue

Draw bar for each time of execution

Listing 4.7:- *Algorithm for runAlgorithm module*

4.2 Integrated Representation of Modules

Evaluation Model is an integration model of all the modules associating for the construction of the simulator. All these modules described in above section are integrated to corporate as a whole working unit.

Tasks are defined with their respective statistical parameters. Task set is a group of tasks that are computed once. Task set, thus formed, is a computational model which gets computed according to the nature of the algorithm. The general outcome of the simulator is the result of the effect of all the modules that constitute the simulator.

The simulator synthesizes the effect of the modules according the task set for particular scheduling algorithms. The general composition of the simulator is illustrated below.

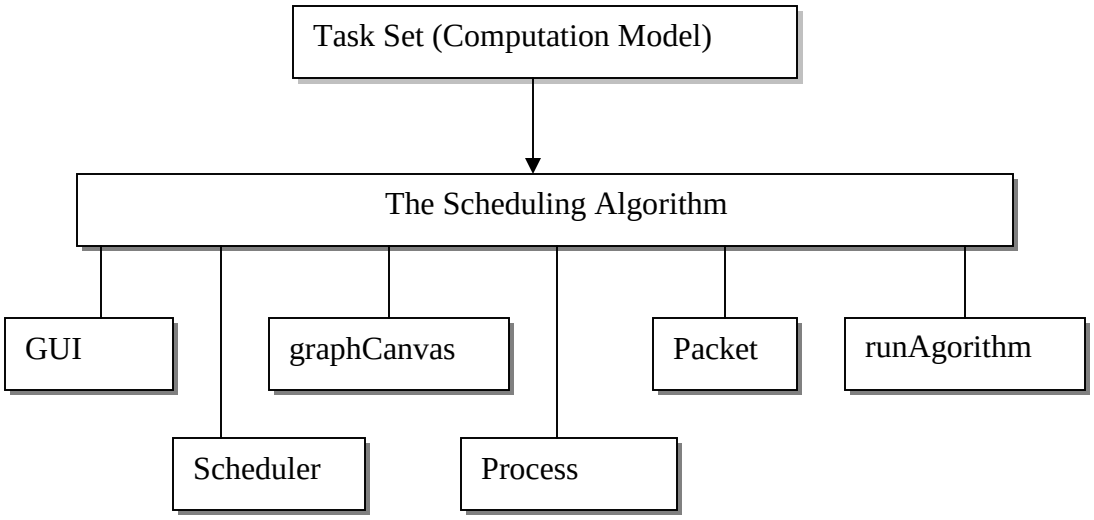

Figure 4.1*: Pictorial representation the modules of the simulator*.

4.3 A deterministic model

The Evaluation Model is a deterministic model reacting to events and generating actions. The waiting time to a detected event is bounded in this model. The action (or actions) taken in waiting to an event is known a priori. A deterministic model implies that each module of the system must have a deterministic behavior that contributes to the overall determinism of the system [34]. The term deterministic describes RTOSes with predictable behavior, in which the completion of operating system calls occurs within known timeframes [7].

CHAPTER V

Implementation

This chapter describes the implementation detail of the Evaluation Model. Since the dissertation is based on the analytical result of the scheduling algorithms, as described in the previous chapters, this chapter visualizes the implementation sketch of the simulator. This simulator is used for making the analytical evaluation of the scheduling algorithms.

.

Section 5.1 describes the tools that are used for the development of the Evaluation Model. These tools take part in the construction of the simulator. Section 5.2 describes about the development model the construction. How exactly the simulator was constructed is explained in this section. To make the simulator easy to comprehend, this section has given the phase-by-phase explanation of the working of the simulator. In section 5.3, inputs and outputs the Evaluation Model operates are described. Inputs and outputs related to the respective scheduling techniques are specified.

5.1 Implementation Tools

These are the constructs that are used for the implementation of the Evaluation Model. As already mentioned, the simulator is a Java applet, these tools are Java programs that facilitate the effective generation of the applet. Handling and management of the Java program files are carried out by these tools.

5.1.1 Javac

This is a java compiler used to compile all the programs implemented to build a simulator. This compiler creates class files (.class extensions) of all java program files (.java files). These class files thus created contain the *bytecode* version of the program. The java *bytecode* is the intermediate representation of the java programs that contains instructions. The output of *javac* is not code that can be directly executed. The Java Virtual Machine will execute the code [10].

5.1.2 Appletviewer

This tool executes the applet in a window. To execute an applet in a web browser, a short HTML text file is written that contains the appropriate APPLET tag. Here are the HTML files that execute applets generated in the dissertation.

Applet for simulating Rate Monotonic Scheduling algorithm

```
<APPLET
        code    = "pdeadlineschedulingrms.class"
        width   = "500"         height  = "300" >
</APPLET>
```

Applet for simulating the Earliest Deadline First algorithm

```
<APPLET
        code    = "pdeadlineschedulingperiodic.class"
                width   = "500"   height= "300" >
</APPLET>
```

Applet for simulating FCFS, RR, STN, SRT algorithms

```
<APPLET
        code    = "pscheduling.class"
                width   = "500"  height= "300" >
</APPLET>
```

5.1.3 Data Structures

Data Structures used in this simulator are Java built-in objects of the Java programming language.

Vector: It is a collection for storing and manipulating the objects stored in it. In this dissertation, the use of vector is done for queuing up the tasks. The tasks are queued for ready state, blocked state and executed state.

String: This is an array of characters. This data structure is used to name the tasks and statistics. All the data manipulation is done with the help of this data structure.

5.2 Evaluation Model development

This section explains the phases of the development of the model. The simulator is constructed by phase-to-phase evolution. Since the dissertation needs to make a deterministic structure of the model for the appropriate follow-up of inputs and outputs of the data taking part in scheduling process, it requires the evolutionary modeling of the system. The evolutionary model is designed in the phase-to-phase way. Though, the section describes the logical overview of the development model, the Evaluation Model is implemented following this approach.

5.2.1 Phase 1

Input Determination and Schedulability Test

Input determination ensures the validation and completeness of the input parameters that are passed to the simulator. *Task set*, which is input to the scheduler, has *number of tasks*, and input parameters like *arrival time, service time, period (deadline)*. Number of tasks and their respective number of parameters, that satisfy the task set, are checked in this phase. If there is some mismatch, warning is generated. This input determination is done according to the type of the scheduling algorithm.

The schedulability for the task set determines whether the tasks in the *task set* meet all the criteria for the scheduling. In Rate Monotonic Scheduling, the schedulability ensures that the tasks do not overcome the *utilization bound* for particular number of tasks in *task set*. The schedulability is evaluated with the help of *utilization bound*.

In Earliest deadline First Scheduling, the schedulability ensures that the tasks do not meet their deadlines. The schedulability is evaluated with the help of *total utilization* of the tasks in the *task set*.

Other scheduling techniques need not test the schedulability for their tasks.

The input parameters are

Task Name (T_1, T_2, ...,Tn)

Arrival Time (A)

Service Time (S)

Completion deadline (D)

Period (P)

5.2.2 Phase 2

Algorithm Execution

In this phase, execution of the scheduling algorithm is done. The tasks in *task set* are scheduled according to the scheduling algorithm, with respect to their predefined input parameters. This phase sets the characteristics of scheduling algorithms to be executed. This is an important phase for developing the simulator since the scheduling algorithms that are being used as part of the analysis needs to be executed in precise way.

a) Rate Monotonic Scheduling:

This algorithm first sorts the periods of the tasks. Smallest the period, highest priority to it is assigned. Task priorities (P) are set according to task periods (Tp), as follows:

$$P = 1/T_p$$

Highest priority task is executed first. Execution of the task is equal to the service time. This execution may occur any time within the duration of period. When this period ends, next period of the task comes to the processor to execute. Each period of the particular task that consumes the processor can be said as the instance of the task for that particular period. Since this model is non-preemptive, the tasks with higher priority do not preempt the tasks with lower priority. In this scheduling, priority is *fixed or static*.

b) Earliest Deadline First:

EDF is a deadline based scheduling technique. In the scheduling, deadlines of the tasks are first sorted and task with nearest current deadline is executed first. Unlike RMS, this is dynamic scheduling technique, meaning the priority changes as the execution of the *task set* continues. The deadline is the absolute deadline, meaning the deadline of the task is calculated from the *arrival time*. As execution goes on, the priority of task may change because of the change in absolute deadline as other tasks get executed. The absolute deadline will be lengthened when the task is being preempted by other task with higher priority. Hence, the priority is *dynamic.*

c) First Come First Serve:

In this scheduling technique, *arrival time* of the task set is sorted and the task with least *arrival time* is executed first. This scheduling has also, thus, fixed priority for the execution of its tasks. The scheduling is carried out by executing task that came to processor earlier than other tasks. When the current executing task is finished, then next task is executed.

d) Round-Robin:

This scheduling technique has priority as that of FCFS, but it uses the time *quantum* for the execution of its tasks. Each task are assigned fixed *quantum* of time to execute and when its *quantum* finishes next task is executed for the same quantum. After this *quantum* time has elapsed, the task is *preempted* and added to the end of the ready queue. When all the tasks are executed with the *quantum time*, the next occurrence of execution with the same manner is done.

e) Shortest Remaining Time:

This scheduling technique assigns priorities to tasks according to the remaining time of the execution of the task. The task which will complete its execution first will be scheduled for execution. The scheduling also uses the dynamic calculation for assigning the priority. The calculation of the *service time* and *arrival time* will give the remaining time of the tasks in comparison to other tasks. The tasks with shorter remaining time will *preempt* the task.

f) Shortest Task Next:

This scheduling technique has the priority according to the duration of task to execute. The shortest task has the highest priority and accordingly priorities for other tasks are assigned. Shortest task is calculated by the time-span of service *time*.

5.2.3 Phase 3

Status Tracing

Status of the scheduling is defined as the events that are occurred till the point of time. This phase gives the exact behavior of the scheduler when it is executing the particular scheduling algorithm. Status tracing manifests the time-to-time execution step of all the tasks in *task set*. This phase gives the precise and clear information about the tasks' execution. It reveals the ready state, serving state, end state of all the tasks that are

currently executing in the simulator with the unit time of execution of the tasks. The unit time can be assumed as the unit executing clock time of the processor.

5.2.4 Phase 4

TimeLine coverage

This phase animates the drawing of the execution of the tasks as timeline. It gives the pictorial description of the execution of each task. The complete eventual description of the scheduling in a timeline manner is shown. This phase has motto to ease the user interface and understanding the scheduling behavior.

5.3 Inputs and Outputs

The algorithms have a specific format of their input and output. In this subsection, the layout of the inputs and outputs of the Evaluation Model for particular scheduling algorithms is described. The scheduling algorithms take input as task set; the tasks with arrival time, service time, period. The scheduler, then, schedules these tasks according to the algorithm. Calculations regarding the algorithms are made in necessary. The outputs of the Evaluation Model are statistics and status of the algorithm execution.

5.3.1 Rate Monotonic Scheduling algorithm

Input:

Task Set: Task (T), Arrival Time (A), Service time(C), Period (P)

Evaluation:

Rate Monotonic Scheduler:

- Utilization Bound Calculation
- Schedulability Testing
- Execution of the algorithm

Output:

Statistics:

- Finish Time, Turnaround Time, Waiting Time and Context Switches

Status:

Algorithm traces (step-by-step execution).

5.3.2 Earliest Deadline First algorithm

Input:

Task Set: Task (T), Arrival Time (A), Service time(C), Completion Deadline (D)

Evaluation:

Earliest Deadline First Scheduler:

Schedulability Testing

Execution of the algorithm

Output:

Statistics:

Finish Time, Turnaround Time, Waiting Time and Context Switches

Status:

Algorithm traces (step-by-step execution).

5.3.3 Other scheduling algorithms

a) First Come First Serve (FCFS)

b) Shortest Task Next (STN)

c) Shortest Remaining Time (SRT)

d) Round Robin (RR)

Input:

Task Set: Task (T), Arrival Time (A), Service time(C)

Evaluation:

Scheduler:

Execution of the algorithm

Output:

Statistics:

Finish Time, Turnaround Time, Waiting Time and Context Switches

Status:

Algorithm traces (step-by-step execution).

CHAPTER VI

Analysis and Experimentation

This chapter describes the way analysis has been performed for the evaluation of prescribed scheduling techniques. The experimental analysis is carried out with the help of the simulator. The working and composition of the simulator is explained in previous chapters. This chapter reveals the exact manipulation of the statistics made in order to accomplish the analytical behavior of the dissertation.

In Section 6.1, the description about the task set analysis for all the scheduling algorithms is given. The task sets are defined manually and experimental analysis is done for particular task set. Description about given input statistics and computed output statistics with respect to the task set is shown for each prescribed scheduling algorithm. The analytical comparison between RMS and EDF is done for the case when only their analysis is relevant and overall analytical comparison is done for the case when the comparison should be done among all algorithms. Section 6.2 describes the processor utilization analysis, since the timing constraints matters only with RMS and EDF, processor utilization is done between only these algorithms. Section 6.3 describes the average turnaround time analysis for all the algorithms for a particular task set. The average is taken from the tasks of the task set. Section 6.4 describes the context switch analysis of all the algorithms. Context switches create runtime overhead so their analysis is crucial. Section 6.5 describes the average waiting time analysis for all the algorithms for a particular task set. Lastly in section 6.6, the results that are inferred from the analytical study are given.

6.1 Task Set Analysis

Task sets are named separately, each with different number of tasks and different input parameters. The result of the output statistics is observed. Finish time, Turnaround time and Waiting time is computed for each task in the task set. Processor utilization concerns with *RMS* and *EDF*, so the computation is made only on those scheduling. This analysis reveals the description of the scheduling phenomenon occurring for a particular task set.

6.1.1 Rate Monotonic Scheduling Vs Earliest Deadline First Scheduling

6.1.1.1 Rate Monotonic Scheduling analysis

Task Set 1:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	2	10	2	2	0
T(2)	1	4	15	7	6	2
T(3)	2	1	11	3	1	0

***Table 6.1**:- Analysis of Rate Monotonic Scheduling for Task Set 1.*

The Total Utilization is: 0.5575757575757576

Execution of Rate Monotonic Scheduling for Task Set 1 (TimeLine Trace):

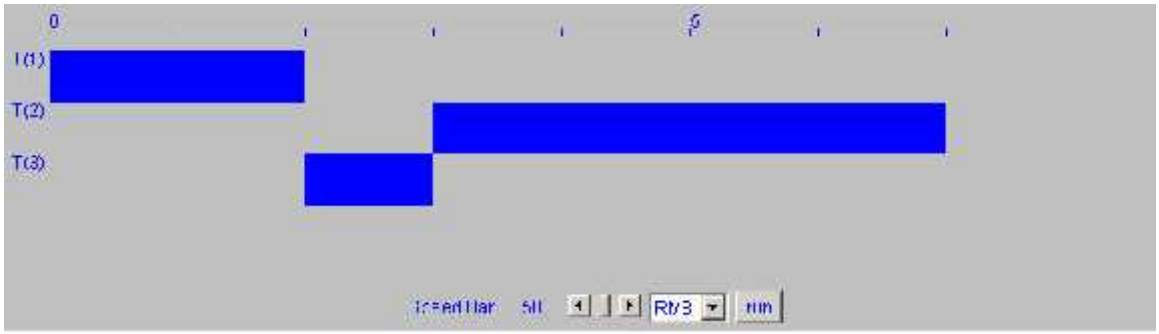

***Fig 6.1:-** Execution of RMS for Task Set 1*

Task Set 2:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	10	80	10	10	0
T(2)	1	30	150	66	65	35
T(3)	2	11	63	21	19	8

T(4)	1	15	120	36	35	20

Table 6.2 *Analysis of Rate monotonic Scheduling for Task Set 2.*

The Total Utilization is: 0.6246031746031746

Execution tracing of Rate Monotonic Scheduling for Task Set 2(TimeLine Trace):

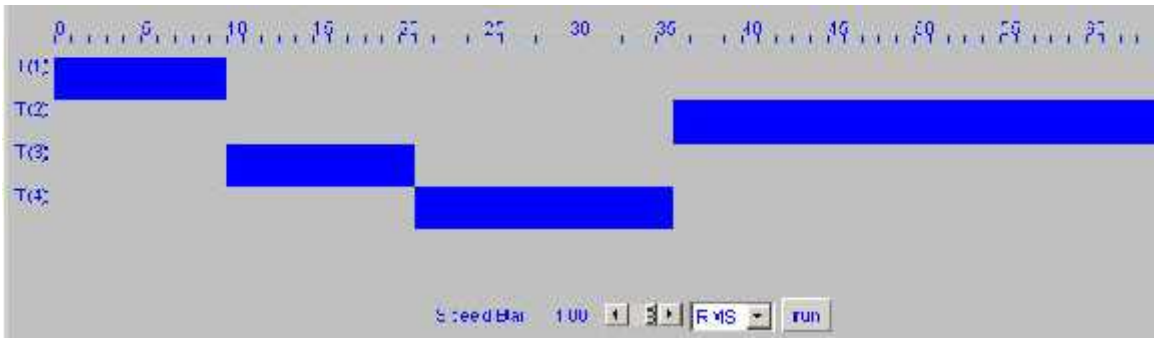

Fig 6.2:- *Execution of RMS for Task Set 2*

Task Set 3:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	2	50	2	2	0
T(2)	0	4	70	6	6	2
T(3)	0	1	110	15	15	14
T(4)	0	5	90	11	11	6
T(5)	0	3	100	14	14	11

Table 6.3*:- Analysis of Rate monotonic Scheduling for Task Set 3.*

The Total Utilization is: 0.1917893217893218

Execution tracing of Rate Monotonic Scheduling for Task Set 3 (TimeLine Trace):

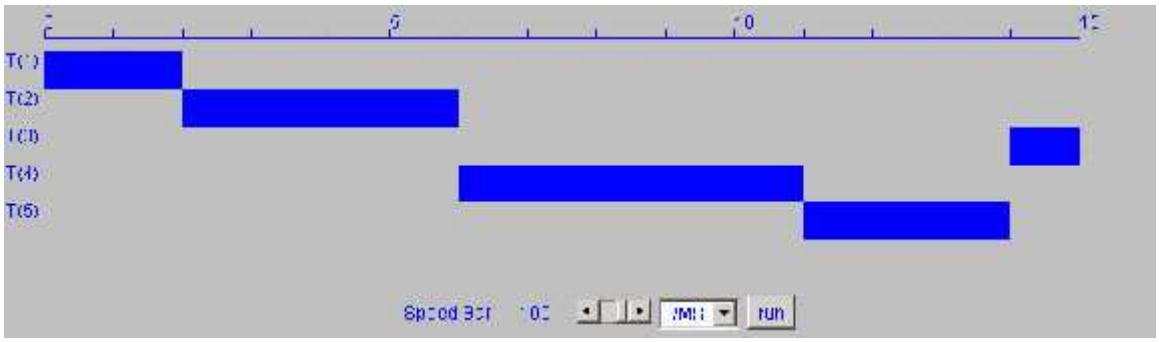

Fig 6.3:- *Execution of RMS for Task Set 3*

Task Set 4:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	20	120	20	20	0
T(2)	1	25	170	80	79	54
T(3)	5	15	150	55	50	35
T(4)	4	10	130	40	36	26
T(5)	6	10	100	30	24	14

Table 6.4*:- Analysis of Rate monotonic Scheduling for Task Set 4.*

The Total Utilization is: 0.5906485671191554

Execution tracing of Rate Monotonic Scheduling for Task Set 4 (TimeLine Trace):

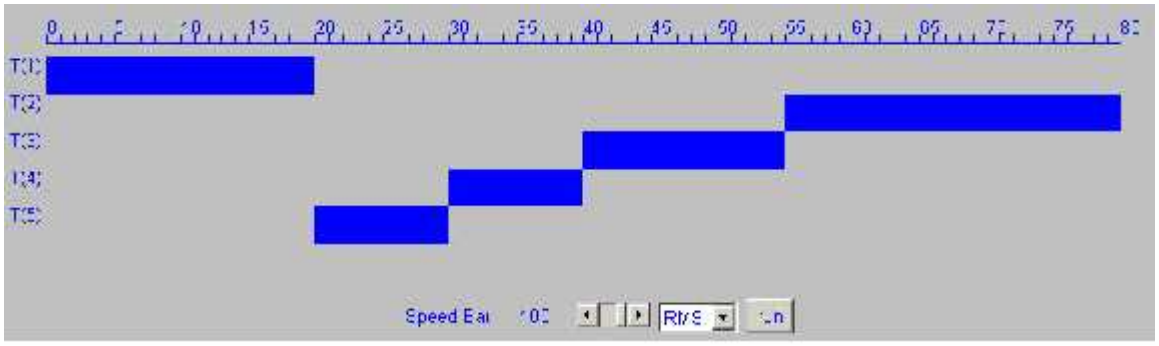

Fig 6.4:- *Execution of RMS for Task Set 4*

Task Set 5:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	7	50			
T(2)	1	9	60			
T(3)	1	15	80			
T(4)	6	5	70			
T(5)	3	10	50			
T(6)	4	11	100			

***Table 6.5**:- Analysis of Rate monotonic Scheduling for Task Set 5.*

The Total Utilization is: 0.8589285714285714

The total utilization required for tasks is greater than the upper bound for RMS. So, RMS can not be used for this task set.

Task Set 6:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	7	50	7	7	0
T(2)	1	9	60	26	25	16
T(3)	1	15	80	37	36	30
T(4)	6	5	70	31	25	20
T(5)	3	10	50	17	14	4
T(6)	4	11	100	41	37	33

***Table 6.6**:- Analysis of Rate Monotonic Scheduling for Task Set 6*

The Total Utilization is: 0.6764285714285716

Execution tracing of Rate Monotonic Scheduling for Task Set 6 (TimeLine Trace):

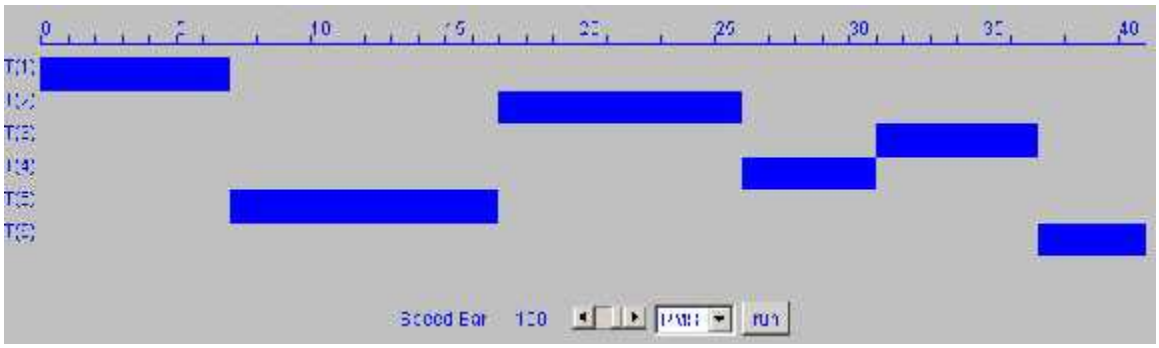

Fig 6.5:- *Execution of RMS for Task Set 6*

Task set 7:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	7	50	7	7	0
T(2)	1	9	60	26	25	16
T(3)	1	15	80	37	36	30
T(4)	6	5	70	31	25	20
T(5)	3	10	50	17	14	4
T(6)	4	11	100	44	40	36
T(7)	2	3	90	40	38	35

Table 6.7*:- Analysis of Rate monotonic Scheduling for Task Set 7*

The Total Utilization is: 0.7097619047619049

Execution tracing of Rate Monotonic Scheduling for Task Set 7 (TimeLine Trace):

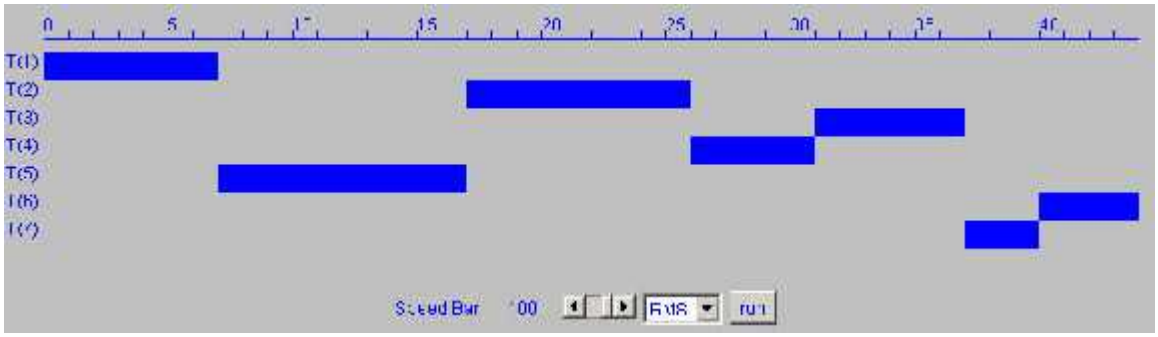

Fig 6.6:- *Execution of RMS for Task Set 7*

6.1.1.2 Earliest Deadline First:

Task Set 1:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	2	10	2	2	0
T(2)	1	4	15	7	6	2
T(3)	2	1	11	3	1	0

Table 6.8:- *Analysis of Earliest Deadline First Scheduling for Task Set1*

The Total Utilization is: 0.5575757575757576

Execution tracing of Earliest Deadline First Scheduling for Task Set 1

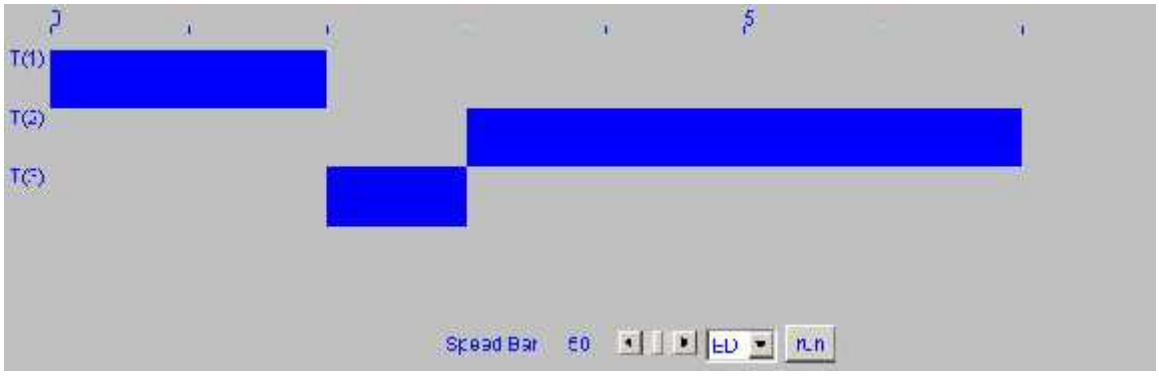

Fig 6.7:- *Execution of EDF for Task Set1*

Task Set 2:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	10	80	13	21	11
T(2)	1	30	150	66	65	35
T(3)	2	11	63	21	11	0
T(4)	1	15	120	36	35	20

***Table 6.9**:- Analysis of Earliest Deadline First Scheduling for Task Set2*

The Total Utilization is: 0.6246031746031746

Execution tracing of Earliest Deadline First Scheduling for Task Set 2

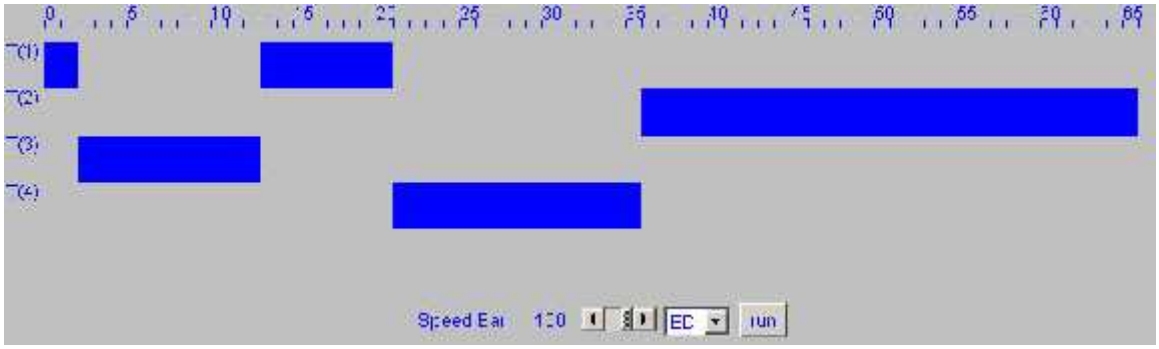

***Fig 6.8:-** Execution of EDF for Task Set 2*

Task Set 3:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	2	50	2	2	0
T(2)	0	4	70	6	6	2
T(3)	0	1	110	15	15	14

T(4)	0	5	90	11	11	6
T(5)	0	3	100	14	14	11

***Table 6.10**:- Analysis of EDF Scheduling for Task Set 3*

The Total Utilization is: 0.1917893217893218

Execution tracing of Earliest Deadline First Scheduling for Task Set 3:

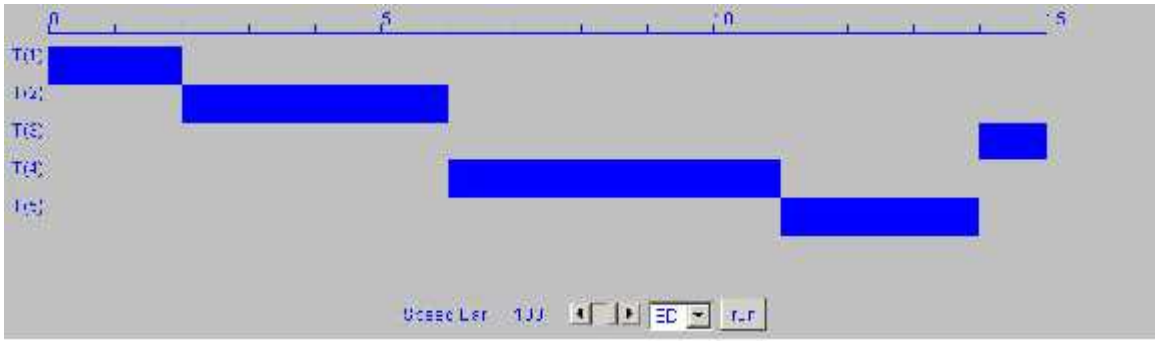

***Fig 6.9:-** Execution of EDF for Task Set 3*

Task Set 4:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	20	120	30	20	0
T(2)	1	25	170	80	79	54
T(3)	5	15	150	55	50	35
T(4)	4	10	130	40	36	26
T(5)	6	10	100	16	24	14

***Table 6.11**:- Analysis of Earliest Deadline First for Task Set 4*

The Total Utilization is: 0.5906485671191554

Execution tracing of Earliest Deadline First Scheduling for Task Set 4

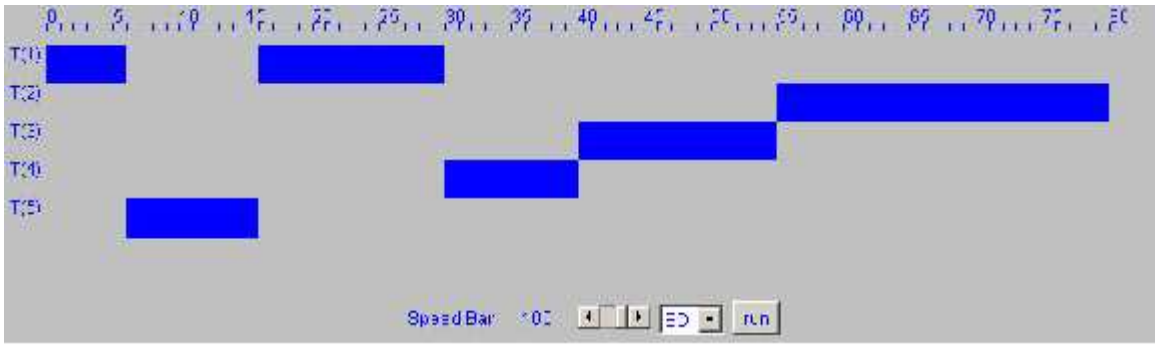

Fig 6.10:- *Execution of EDF for Task Set 4*

Task Set 5:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	7	50	7	7	0
T(2)	1	9	60	26	25	16
T(3)	1	15	80	46	45	30
T(4)	6	5	70	31	25	20
T(5)	3	10	50	17	14	4
T(6)	4	11	100	57	53	42

Table 6.12*:- Analysis of Earliest Deadline First for Task Set 5*

The Total Utilization is: 0.8589285714285714

Execution tracing of Earliest Deadline First Scheduling for Task Set 5:

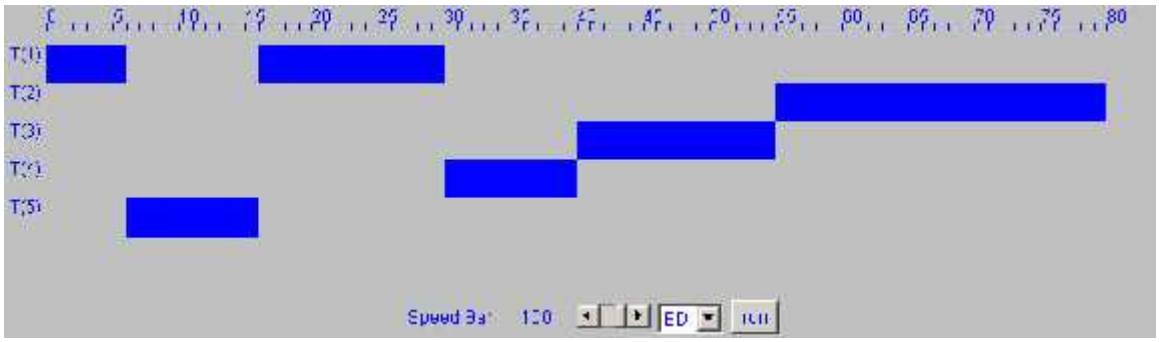

Fig 6.11:- *Execution of EDF for Task Set 5*

Task Set 6:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	7	50	7	7	0
T(2)	1	9	60	26	25	16
T(3)	1	15	80	37	36	30
T(4)	6	5	70	31	25	20
T(5)	3	10	50	17	14	4
T(6)	4	11	100	41	37	33

Table 6.13*:- Analysis of Earliest Deadline First for Task Set 6*

The Total Utilization is: 0.6764285714285716

Execution tracing of Earliest Deadline First Scheduling for Task Set 6:

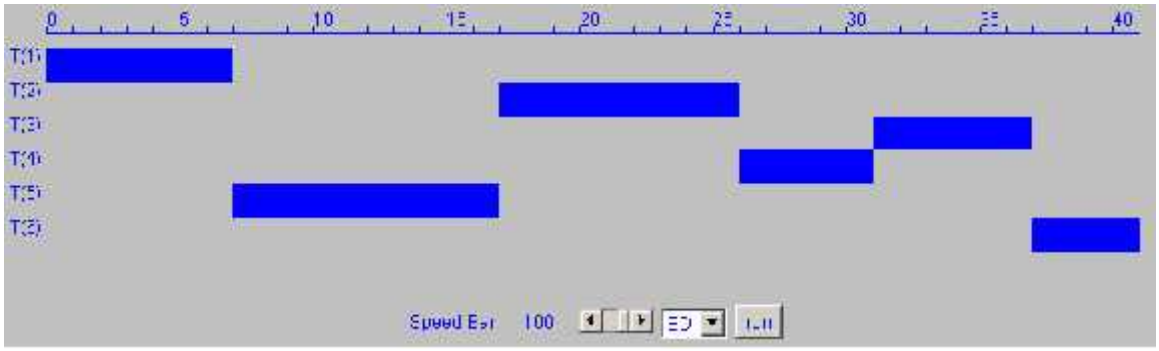

Fig 6.12:- *Execution of EDF for Task Set 6*

Task Set 7:

Task Name	Arrival Time	Service Time	Period	Finish Time	Turn around time	Waiting Time
T(1)	0	7	50	7	7	0
T(2)	1	9	60	26	25	16
T(3)	1	15	80	37	36	30
T(4)	6	5	70	31	25	20
T(5)	3	10	50	17	14	4
T(6)	4	11	100	44	40	36
T(7)	2	3	90	40	38	35

***Table 6.14**:- Analysis of EDF for Task Set 7*

The Total Utilization is: 0.7097619047619049

Execution tracing of Earliest Deadline First Scheduling for Task Set 7:

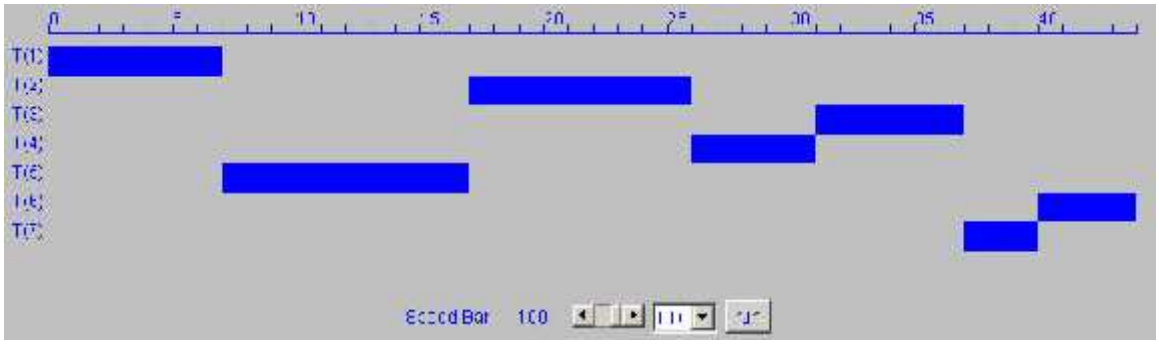

***Fig 6.13:-** Execution of EDF for Task Set 7*

6.1.2 Analysis of General scheduling algorithms

Task Set 1:

First Come First Serve:

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
T(1)	0	2	2	2	0
T(2)	1	4	6	5	1
T(3)	2	1	7	5	4

***Table 6.15**:- Analysis of FCFS for Task Set 1*

Execution tracing of First Come First Serve Algorithm

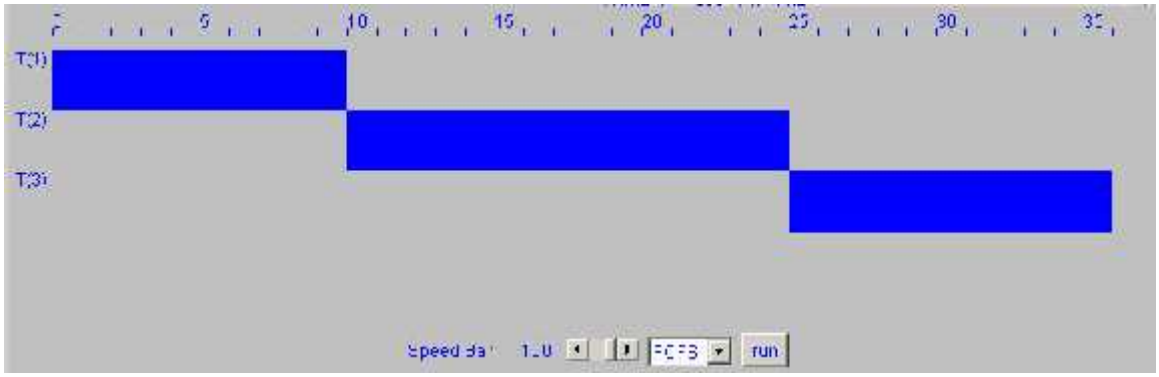

***Fig 6.14:-** Execution of FCFS for Task Set 1*

Round Robin with quantum 1:

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
T(1)	0	2	4	4	2
T(2)	1	4	7	6	2
T(3)	2	1	3	1	0

***Table 6.16**:- Analysis of Round Robin (quantum 1) for Task Set 1*

Execution tracing for Round Robin with quantum 1

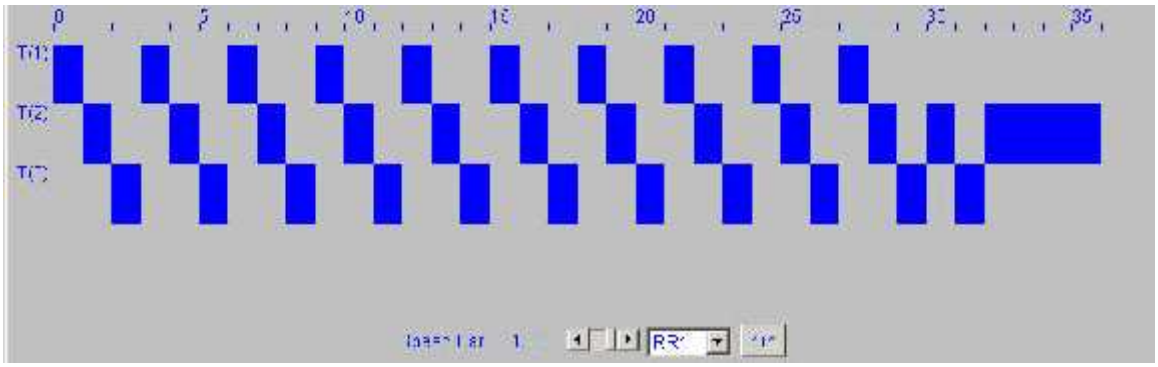

Fig 6.15:- *Execution of RR1 for Task Set 1*

Round Robin with quantum 4

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
T(1)	0	2	2	2	0
T(2)	1	4	6	5	1
T(3)	2	1	7	5	4

Table 6.17*:- Analysis of Round Robin (quantum 4) for Task Set 1*

Execution tracing for Round Robin with quantum 4:

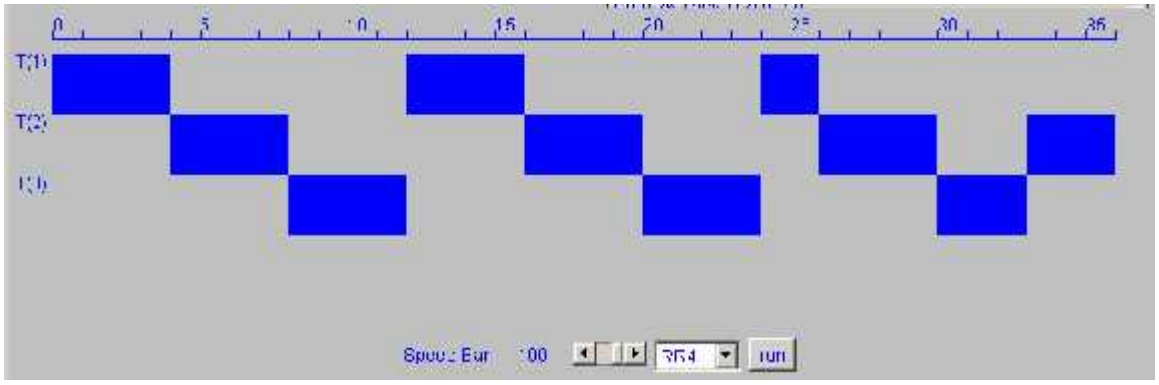

Fig 6.16:- *Execution of RR4 for Task Set1*

Shortest Task Next:

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
T(1)	0	2	2	2	0
T(2)	1	4	7	6	2
T(3)	2	1	3	1	0

***Table 6.18**:- Analysis of Shortest Task Next for Task Set 1*

Execution tracing for Shortest Task Next scheduling:

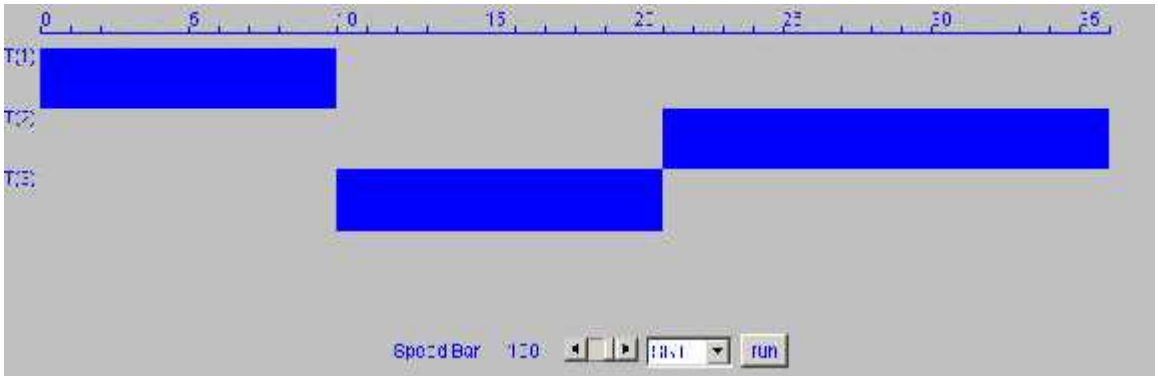

***Fig 6.17:-** Execution of STN for Task Set1*

Shortest Remaining Time Next:

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
T(1)	0	2	2	2	0
T(2)	1	4	7	6	2
T(3)	2	1	3	1	0

***Table 6.19:-** Analysis of Shortest Remaining Time Next for Task Set 1*

Execution tracing for SRT

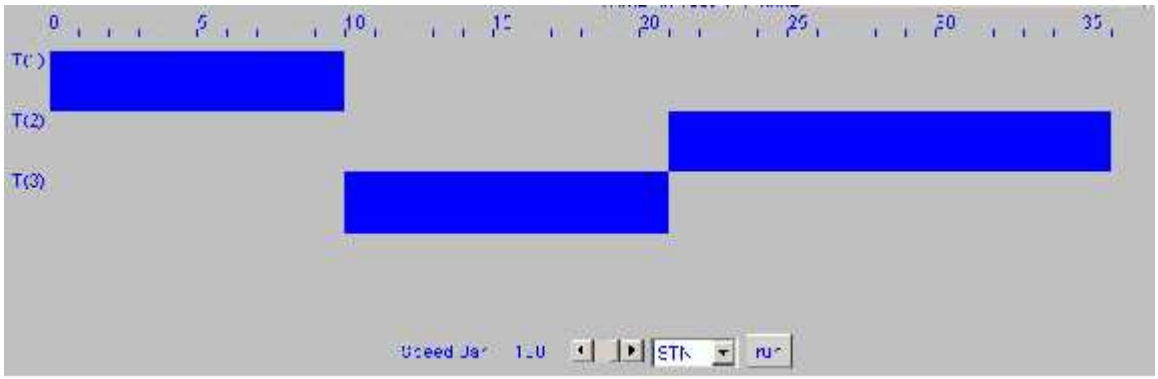

Fig 6.18:- *Execution of SRT for Task Set1*

TASK SET 2:

First Come First Serve:

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
1	0	3	3	3	0
T(2)	2	6	9	7	1
T(3)	4	4	13	9	5
T(4)	6	5	18	12	7
T(5)	8	2	20	12	10

Table 6.20*:- Analysis of FCFS for Task Set 8*

Execution tracing for FCFS

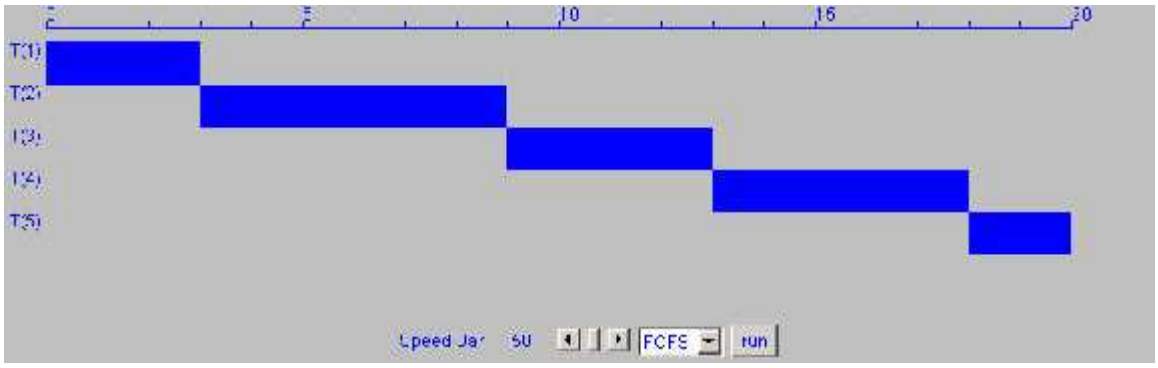

Fig 6.19:- *Execution of FCFS for Task Set 8*

Round Robin with quantum 1

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
1	0	3	4	4	1
T(2)	2	6	19	17	11
T(3)	4	4	17	13	9
T(4)	6	5	20	14	9
T(5)	8	2	15	7	5

***Table 6.21**:- Analysis of Round Robin with quantum 1 for Task Set 8*

Execution tracing for RR 1

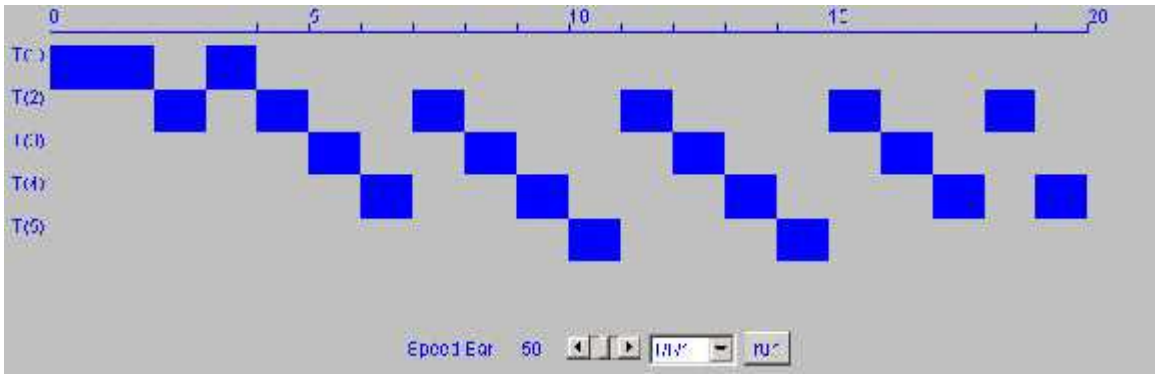

***Fig 6.20:-** Execution of RR1 for Task Set 8*

Round Robin with quantum 4

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
1	0	3	3	3	0
T(2)	2	6	19	17	11
T(3)	4	4	11	7	3
T(4)	6	5	20	14	9

T(5)	8	2	17	9	7

***Table 6.22**:- Analysis of Round Robin with quantum 4 for Task Set 8*

Execution tracing for RR 4

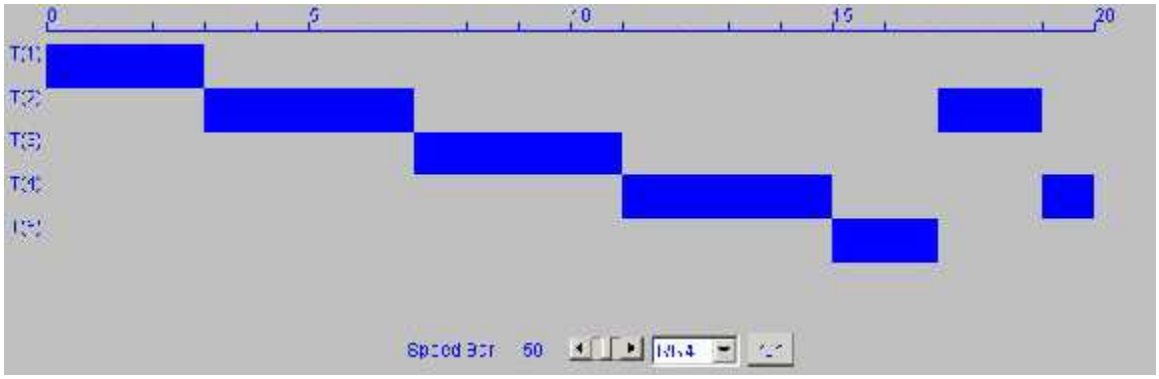

***Fig 6.21:-** Execution of RR4 for Task Set 8*

Shortest Task Next:

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
1	0	3	3	3	0
T(2)	2	6	9	7	1
T(3)	4	4	15	11	7
T(4)	6	5	20	14	9
T(5)	8	2	11	5	3

***Table 6.23**:- Analysis of Shortest Task Next for Task Set 8*

Execution tracing for STN:

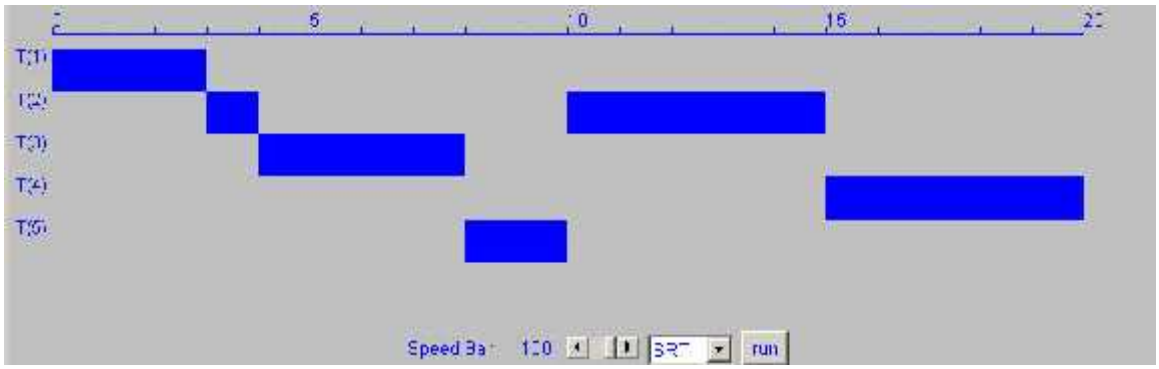

Fig 6.22:- *Execution of STN for Task Set 8*

Shortest Remaining Time Next:

Task Name	Arrival Time	Service Time	Finish Time	Turn around time	Waiting Time
1	0	3	3	3	0
T(2)	2	6	19	17	11
T(3)	4	4	11	7	3
T(4)	6	5	20	14	9
T(5)	8	2	17	9	7

Table 6.24:- *Analysis of Shortest Remaining Time Next for Task Set 8*

Execution tracing for SRT:

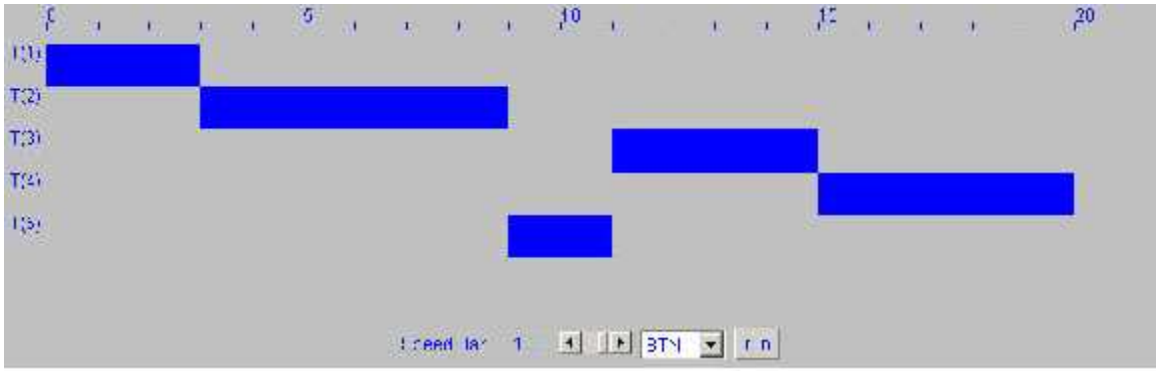

Fig 6.23:- *Execution of SRT for Task Set 8*

6.2 Processor Utilization Analysis

As stated above, the computation of utilization is made on RMS and EDF. This is because only these scheduling techniques deal with the deadlines of the tasks and the utilization of the tasks concerns with meeting of those deadlines.

Rate Monotonic scheduling Vs Earliest Deadline First

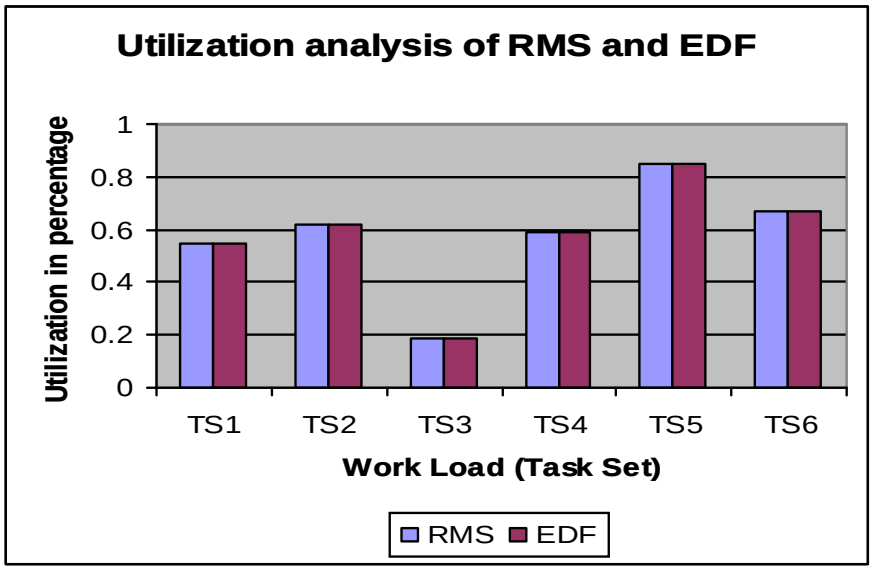

Chart 6.1:- *Utilization analysis of RMS and EDF*

6.3 Average Turnaround Time Analysis

Computation of average turnaround time is done by taking average of the turnaround time of all tasks in the task set. Comparison is made between the scheduling techniques for respective task sets.

6.3.1 Rate Monotonic Vs Earliest Deadline First

Analysis of average turnaround time between RMS and EDF will help to conclude their feasibility. Computation of the average turnaround time for different tasks sets is made and compared with the two scheduling techniques.

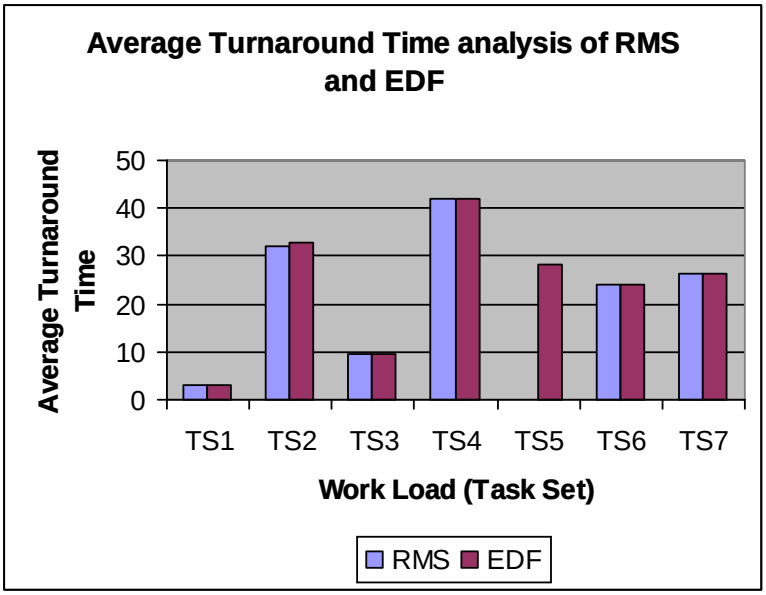

Chart 6.2:- *Average Turnaround Time analysis of RMS and EDF*

6.3.2 Overall analysis of the scheduling algorithms

The computation of the average turnaround time is made for particular task set and comparison is done between RMS, EDF and general scheduling techniques. This analysis will help to determine the feasibility of all scheduling techniques. The general scheduling techniques can be analyzed as the candidate scheduling techniques for the Real-Time systems.

TASK SET 1:

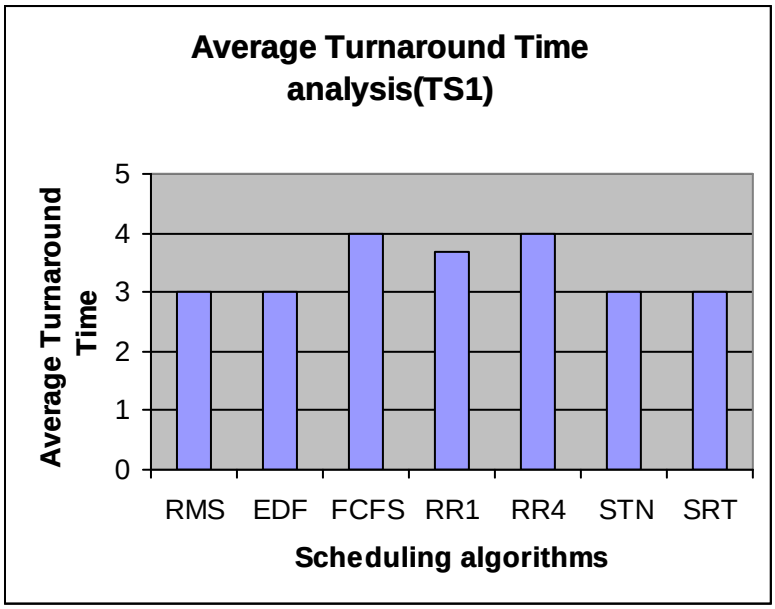

Chart 6.3:- *Average Turnaround Time analysis for task set 1*

TASK SET 7:

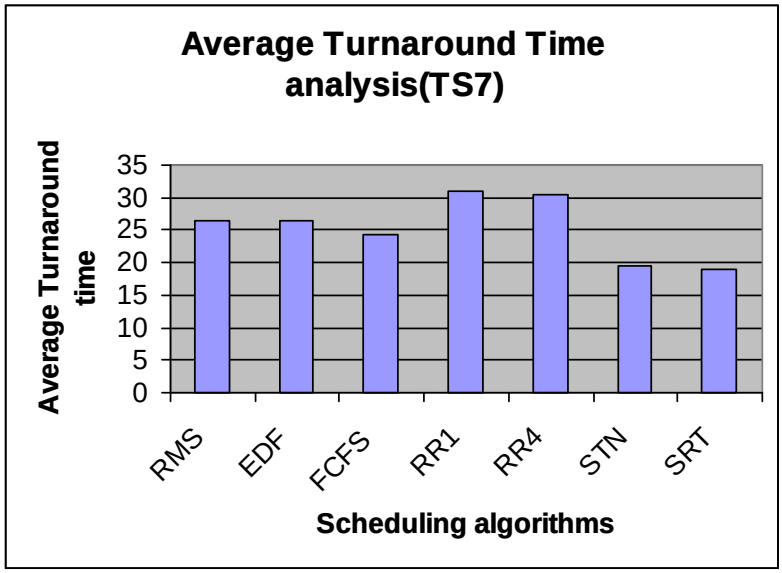

Chart 7.4:- *Average Turnaround Time analysis for task set 7*

6.4 Context Switches Analysis

Computation of the context switches is made for particular task set and comparison is made to all scheduling techniques. Context switches are accounted when the new task is loaded or the task gets preempted. Context switch analysis is a key point for deciding the overhead generated by the scheduling scheme. Following are the evalulations made for the given task sets. Analytical comparison is made to all scheduling algorithms for the task set.

TASK SET 4:

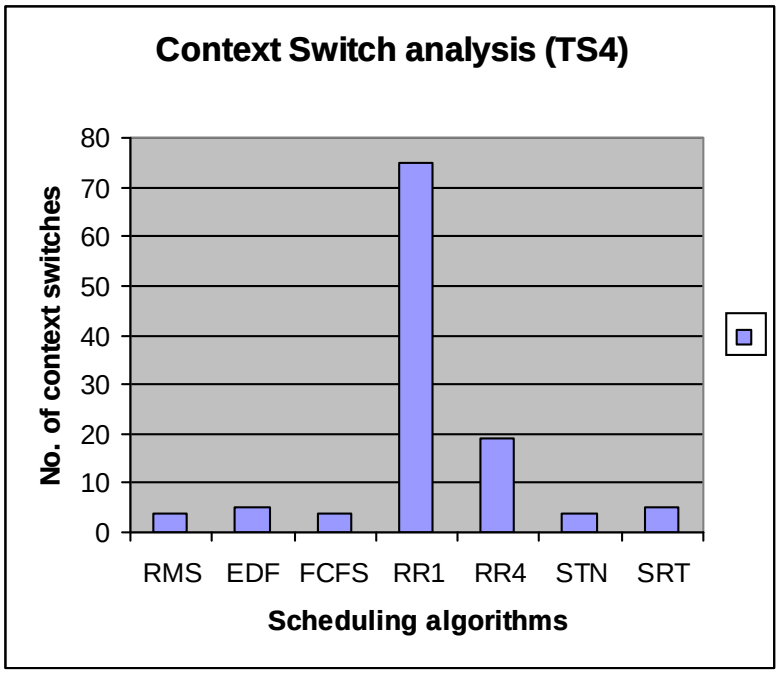

Chart 6.5:- *Context Switch analysis for task set 4*

TASK SET 7:

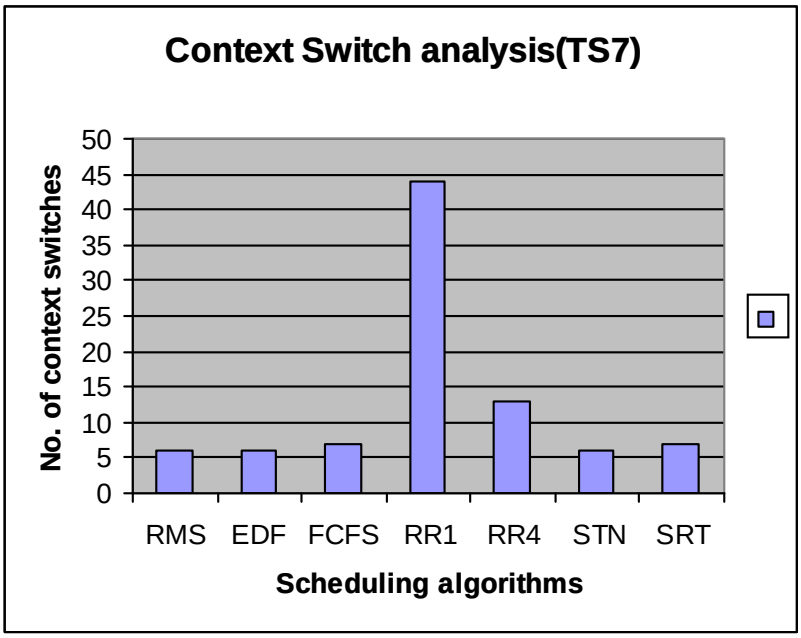

Chart 6.6:- *Context Switch analysis for task set 7*

6.5 Average Waiting Time Analysis

The computation of the average waiting time is made for particular task set and comparison is done between RMS, EDF and general scheduling techniques. This analysis will help to calculate the timely response of all scheduling techniques. Following are the evaluations made for the given task sets.

TASK SET 4:

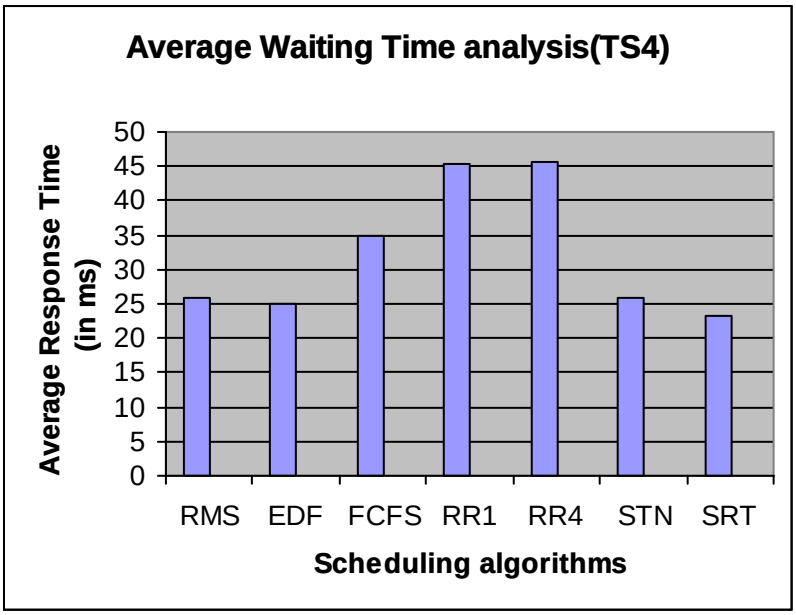

Chart 6.7:- *Average Waiting Time analysis for task set 4*

TASK SET 7:

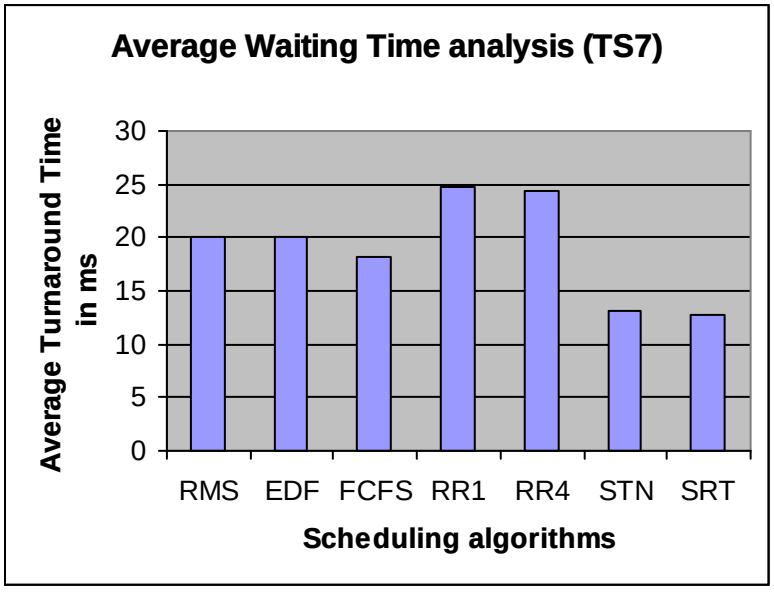

Chart 6.8:- *Average Waiting Time analysis for task set 7*

6.6 Results

From the analysis, both EDF and RMS meet all deadlines for small values of Utilization. For higher values of Utilization (above 90%), RMS starts to see some deadline misses. This is very much in accordance with RMS theory. Utilization lower bound for RMS is about 69 percent for an arbitrarily large number of tasks. A task set with U less than 69% should be able to meet all of its deadlines. For higher values of Utilization, the task set may or may not be schedulable under RMS. This is verified by the simulator analysis.

EDF should be able to schedule all task sets with U less than 100 percent. We observe that this is more or less true. All deadlines are satisfied for Utilization as high as 98 percent. Since the theory does not account for the time spent executing the RTOS and since it is not possible to schedule a task set with U = 100% utilization in reality, some deadline misses are seen as Utilization become exactly 100 percent. An interesting observation here is the number of deadline misses for U>100 percent. EDF does a poor job of scheduling tasks in overload situation. It schedules the task that is closest to missing its deadline in overload situation and therefore creates a cascade of deadline misses. RMS performs better in than EDF at this time.

The empirical analysis motivates Rate Monotonic Scheduling (RMS) as the efficiently applicable fixed-priority scheduling technique for Real-Time systems with limited utilization. If utilization is conciliated then RMS could beat all the candidate scheduling techniques. The RMS studied in this thesis has less context-switches, the runtime overhead is less which is a required factor for any Real-Time applications. The analysis shows that it has the spectacular overcome to EDF and other prescribed scheduling algorithms for the feasibility point of view when the number of tasks is less.
The optimality of the RMS is the consequence of the less runtime overhead, less average turnaround time, less waiting time with compromised utilization and higher predictability.

CHAPTER VII

7.1 Conclusion

In this dissertation, the comparison of the behavior of the two most famous scheduling policies used today for developing real-time applications: RMS and EDF, has been done. Although widely used, in fact, there are still many misconceptions about the properties of these two scheduling methods, mainly concerning their implementation complexity, the runtime overhead and the behavior they introduce.

The evaluation is done with the help of an Evaluation Model, a simulator, which is constructed by coding in Java Language. The bulk of time is consumed by its construction in order to accomplish this dissertation.

The analytical result of the above experimentation infers the optimal advantage of RMS over EDF is its simpler implementation in commercial kernels (RTOS) because of its low context switch overhead. The average turnaround time is also less than EDF.

On the other hand, EDF allows the full processor utilization, which implies a more efficient exploitation of computational resources and a much better responsiveness. These properties become very important for embedded systems working with limited computational resources, and for multimedia systems, where quality of service is controlled through resource reservation mechanisms that are much more efficient under EDF. The study shows the average waiting time is less than RMS.

Rate Monotonic Scheduling and Earliest Deadline First scheduling are the only feasible algorithms for real time systems since they compute the time constraints. Satisfying the timing constraints is the foremost concern of the Real-Time systems. Only shortcoming these scheduling algorithms divulge is the restricted utilization. This restricted utilization can negotiate with the likelihood of their use for Real-Time applications.

Other scheduling algorithms used in this dissertation were put forward to be candidate for real-time scheduling. The analysis shows that they do not suit for application in real-time

systems scheduling since they do not comprise time constraints. They have nothing to do with *deadlines*. Though, the evaluation manifests the following results:

FCFS: FIFO queue, simple, fair, but poor performance high average turnaround time.
STN: Optimal for minimizing queuing time, low average turnaround time. It tries to predict the task to schedule based on their previous history.
RR: Depends on choosing the quantum

Too short - Too many context-switches leads to runtime overhead.

Too long - reschedule latency is too great.

If many tasks want the CPU, then it's a long time before a particular task can get the CPU. This then acts like FCFS.
SRT: This algorithm performs like EDF without time constraints, good average turnaround time, better average response time for higher number of tasks.

Understanding the basics of real-time theory is therefore of fundamental importance to architect the Real-Time systems. The reliability and the predictability of the systems is sustained by the scheduling policy in the system.

7.2 Further Recommendation

Most of the scheduling techniques described in this thesis are based on uniprocessor system. But they are equally applicable to multiprocessor/distributed systems, if we assume that there exists an appropriate allocation scheme which statically allocates tasks to various processors, and the scheduling is done on each processor independently using the uniprocessor techniques discussed in this thesis.

The study of analytical evaluation of priority-based scheduling algorithms will lead to their application to Fault-Tolerance applications. The general approach to fault tolerance is to maintain enough slack in the schedule so that any task instance can be re-executed if a fault occurs during its execution. Tasks are executed following the usual RMS or EDF scheme if no faults occur. However, if a fault occurs in a task, a recovery scheme can be used to re-execute the task.

References

[1] Devillers R., Goossens J., *Liu and Layaland's Schedulability Test Revisited*, December 28, 1998.

[2] Liu C.L., Layland, J.W., *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,* 1973.

[3] Mohammadi A., Akl S. G., *Scheduling algorithms for Real-Time Systems*, Queen's University, Canada, July 15, 2005.

[4] Balarin F., Lavagno L., Murthy P., Sangiovanni-Vincentelli A., *Scheduling for Embedded Real-Time Systems,* University of California at Berkeley, 1998.

[5] Baruah S., Goossens J., *Scheduling Real-time Tasks: Algorithms and Complexity*, University of North Carolina at ChapelHill, University of Librede Bruxelles.

[6] Koch B., *The Theory of Task Scheduling in Real-Time Systems*, Hamburg University, December 1999.

[7] Clifford W. M., *An Introduction to Real-Time Operating Systems: Scheduling Theory,* School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 13, 1992.

[8] Cottet F., Kaiser C., Mammeri Z., *Scheduling in Real-Time Systems*, Wiley Publications, 2004.

[9] Leung J. and Zhao H., *Real-Time Scheduling Analysis*, Department of Computer Science, New Jersey , Institute of Technology, November 2005.

[10] Nilsson R., *Automated Timeliness Testing of Dynamic Real-Time systems*, April 21, 2003.

[11] Tanenbaum A. S., *Modern Operating Systems*, Second Edition, Prentice Hall of India, 2003.

[12] Stankovic J. A., *Misconceptions about Real-Time Computing*, University of Massachusetts, 1987.

[13] Obenza R., *Rate Monotonic Analysis for Real-Time Systems*, University of Virginia, 1993.

[14] Bruda S. D., Akl S.G., *Real–Time Computation: Formal Definition and its Applications*, Queen's University, 2000.

[15] Anvari M., *Real-Time Scheduling – An Overview*, 2000.

[16] Audsley N., Burns A, *Real-Time System Scheduling*, University of York, UK, 2004.

[17] Panzieri F., Davoli R., *Real–Time Systems: A Tutorial*, University of Bologna (Italy), 1993.

[18] Kamal R., *Embedded Systems, Architecture, Programming and Design*, Tata Mc-Graw Hill Publishingh Company Ltd, reprint 2003.

[19] Gambier A., *Real-time Control Systems: A Tutorial*, University of Mannheim, 68131 Mannheim, Germany, 1996.

[20] Li Q., Yao C., *Real-time Concepts for Embedded Systems*, Published by CMP Books, CMP Media LLC.

[21] Thoen F., Catthoor F., Verkest D. Wong C., *Requirements for Static Task Scheduling in Real Time Embedded Systems*, IMEC, Kapeldreef 75, B-3001, Leuven, Belgium, 2001.

[22] Ramamritham K. , Stankovic J. A., *Scheduling Algorithms and Operating Systems Support for Real-Time Systems,* University of Massachusetts, Amherst, MA 01003, 1998.

[23] Buckl C., *Developing dependable real-time systems*, University of Germany, 2006.

[24] Ghosh S., Ph.D, *Guaranteeing Fault-Tolerance through Scheduling in Real-Time Systems,* University of Pittsburgh, 1996.

[25] Audsley N., Burns A., Richardson M., Tindell K., Wellings A.J., *Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling,* Department of Computer Science, University of York, England, 1999.

[26] Lauzac S., Melhem R., F., *An Improved Rate-Monotonic Admission Control and Its Applications*, IEEE, and Daniel Mosse´, Member, IEEE Computer Society, 2003.

[27] Lehoczky J., Lui S., Ding Y., *The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior*, Carnegie Mellon University, 1989.

[28] Bin E., Buttazzo G. C., *The Space of Rate Monotonic Schedulability*, 2002.

[29] Caccamo M. and Buttazzo G., Anna S., *Optimal Scheduling for Fault-Tolerant and Firm Real-Time Systems,* Pisa, Italy, 1998.

[30] Sanfridson M., *Quality of control and real-time scheduling*, Department of Machine Design, University of Sweden, 2004.

[31] Cayssials R., Orozco J., Santos J. and Santos R., *Rate Monotonic Scheduling of Real-Time Control Systems with the minimum number of Priority Levels*, Universidad Nacional del Sur / CONICET Avda. Alem 1253, 8000 Bahía Blanca, Argentina, 2001.

[32] Baruah S., Burns A., *Sustainable Scheduling Analysis*, University of North Carolina, University of York, UK, 2006.

[33] Deters M., Gill C., Cytron R., *Rate-Monotonic Analysis in the C++ Type System*, Department of Computer Science and Engineering Washington University in St.Louis, 1994.

[34] Wehrmeister M. A., Becker L. B., Pereira C. E., *Optimizing Real-Time Embedded Systems Development Using a RTSJ-based API*, Federal University of Rio Grande do Sul, Brazil, 2004.

[35] Schildt H., *The Complete Reference Java*, J2SE 5 Edition, Tata McGraw-Hill Edition, 2005.

[36] Hu Y., *A Portable Worst – Case Execution Time Analysis Framework for Real-Time Java Architectures*, Real-Time Systems Research Group Department of Computer Science ,University of York, 2004.

[37] Shuzhen D., Qiwen X. and Naijun Z., *A Formal Proof of the Rate Monotonic Scheduler*, International Institute For Software Technology The United Nations University,1999.

[38] Rushby J., *Critical System Properties: Survey and Taxonomy,* Computer Science Laboratory SRI International Menlo Park CA94025 USA, 1994.

[39] Shin K. G., *Current Status and Future Directions of Real-Time Computing*, University of Michigan, 1993.

[40] Lowe D., *Java™ All-in-One Desk Reference For Dummies,* Published by Wiley Publishing, Inc, pages 12-25, 2004.

[41] Spuri, M. and G.C. Buttazzo, *Scheduling Aperiodic Tasks in Dynamic Priority Systems, Journal of Real-Time Systems,* Vol. 10, No. 2, pages 10-20, 1996.

[42] Punnekkat S., *Schedulability Analysis for Fault Tolerant Real-Time Systems*, University of York, June 1997.

Appendix A

Class graphCanvas

```
imports java.lang.Object
imports java.awt.Component
imports java.awt.Canvas
```

public class **graphCanvas** extends Canvas

constructor

graphCanvas(pdeadlinescheduling) // scheduling algorithm

method

paint(Graphics)

constructors

graphCanvas

```
public graphCanvas(pdeadlinescheduling algorithm)
```

methods

paint

```
public void paint(Graphics g)
```

Overrides:

paint in class Canvas

Appendix B

Class GUI

```
imports java.lang.Object
imports java.util.Observable
```

public class **GUI** extends Observable

Constructor index

GUI()

Method index

input(Object)

Constructor

GUI

```
public GUI()
```

Method

input

public void input(Object info)

Appendix C

Class Packet

```
imports java.lang.Object
imports scheduling.processdeadline.Packet
```

public class **Packet** extends Object

Constructor index

Packet(String, String, String, String, String)

Method index

getAlg()

getArriv()

getProc()

getSdead()

getServ()

constructors

Packet

```
public Packet(String p, String a, String s, String d, String al)
```

Methods

getAlg

```
public String getAlg()
```

getArriv

```
public String getArriv()
```

getProc

```
public String getProc()
```

getSdead

```
public String getSdead()
```

getServ

public String getServ()

Appendix D

Class pdeadlinescheduling

```
imports java.lang.Object
imports java.awt.Component
imports java.awt.Container
imports java.awt.Panel

imports java.applet.Applet
```

public class **pdeadlinescheduling** extends Applet implements ActionListener

Constructor index

pdeadlinescheduling()

Method index

actionPerformed(ActionEvent)

cleargrid()

drawbar(process, int)

getAppletInfo()

Returns information about this applet.

handleEvent(Event)

init()

Initializes the applet.

paint(Graphics)

Paints the applet.

paintBoard(Graphics)

report(Vector, String)

resetGUI()

setgrid(Vector)

setTitle(String)

upstatus(String)

Constructors

pdeadlinescheduling

```
public pdeadlinescheduling()
```

Methods

actionPerformed

```
public void actionPerformed(ActionEvent evt)
```

cleargrid

```
public void cleargrid()
```

drawbar

```
public void drawbar(process P, int t)
```

getAppletInfo

```
public String getAppletInfo()
```

Returns information about this applet.

Returns:

a string of information about this applet

Overrides:

getAppletInfo in class Applet

handleEvent

```
public boolean handleEvent(Event evt)
```

Overrides:

handleEvent in class Component

init

```
public void init()
```

Initializes the applet.

Overrides:

init in class Applet

start, stop, destroy

paint

```
public void paint(Graphics g)
```

Paints the applet. If the applet does not need to be painted (e.g. if it is only a container for other awt components) then this method can be safely removed.

Parameters:

g - the specified Graphics window

Overrides:

paint in class Container

update

paintBoard

```
public void paintBoard(Graphics g)
```

report

```
public void report(Vector R, String title)
```

resetGUI

```
public void resetGUI()
```

setgrid

```
public int setgrid(Vector L)
```

setTitle

```
public void setTitle(String txt)
```

upstatus

public void upstatus(String txt)

Appendix E

Class process

```
imports java.lang.Object
imports scheduling.processdeadline.process
```

public class **process** extends *Object*

Constructor index

process(String, int, int, int, int)

Methods index

getArrival()

getFinish()

getName()

getProcessNum()

getService()

getStartingDeadline()

getTminus()

getTq()

getTqs()

report(int)

servicing()

Constructor

process

```
public process(String n,int a,int s,int processNum,int d)
```

Methods

getArrival

```
public int getArrival()
```

getFinish

```
public int getFinish()
```

getName

```
public String getName()
```

getProcessNum

```
public int getProcessNum()
```

getService

```
public int getService()
```

getStartingDeadline

```
public int getStartingDeadline()
```

getTminus

```
public int getTminus()
```

getTq

```
public double getTq()
```

getTqs

```
public double getTqs()
```

report

```
public void report(int t)
```

servicing

public void servicing()

Appendix F

Class runAlgorithm

```
imports java.lang.Object
imports scheduling.processdeadline.Scheduler

public class runAlgorithm extends Scheduler implements Runnable
```

Constructor index

runAlgorithm(Vector, pdeadlinescheduling, int, String)

Method index

run()

constructors

runAlgorithm

```
public runAlgorithm(Vector q, pdeadlinescheduling i, int c,
                                                String algo)
```

Methods

run

public void run()

Appendix G

Class Scheduler

```
imports java.lang.Object
imports scheduling.processdeadline.Scheduler
```

public class **Scheduler** extends Object

Variable index

thread

Constructor indeces

Scheduler(Vector, pdeadlinescheduling, int)

Scheduler(Vector, pdeadlinescheduling, int, String)

Methods indeces

processready(int)

resetQ()

variable

thread

```
public Thread thread
```

Constructors

Scheduler

```
public Scheduler(Vector q,pdeadlinescheduling i,int c)
```

Scheduler

```
public Scheduler(Vector q,pdeadlinescheduling i,int c,String algo)
```

methods

processready

```
public process processready(int tick)
```

resetQ

public void resetQ()

Appendix H

Class Hierarchy

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Canvas
      - class scheduling.processdeadline.graphCanvas
    - class java.awt.Container
      - class java.awt.Panel
        - class java.applet.Applet
          - class scheduling.processdeadline.pdeadlineschedu ling (implements java.awt.event.ActionListener)
  - class java.util.Observable
    - class scheduling.processdeadline.GUI
  - class scheduling.processdeadline.Packet
  - class scheduling.processdeadline.Scheduler
    - class scheduling.processdeadline.runAlgorithm (implements java.lang.Runnable)
    - class scheduling.processdeadline.process