CHAPTER 1

INTRODUCTION

1.1 Introduction

Real-time computer systems [19] differ from non-real time computer systems in 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. If the external process is simple, then a single microcomputer 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 software structure, and a more sophisticated means of coordination among multiple activities; this motivates the need for methods of scheduling various activities in all but the simplest real-time systems. 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.

To begin our study of real-time systems and strategies for scheduling, we will first concentrate on defining the problem. We use the term “real-time system” or “real-time application” in the broadest sense to refer to the entire system including the physical environment, the software, and the computer hardware. The system is divided into several layers. At the top, we have tasks which are generic computational entities that have timing constraints and synchronization and communication relationships. These tasks are derived by some design process from the abstract timing requirement of the whole physical system. The abstract computational activities or as implementation dependent programs supported by real-time operating system. The time constraints for a task include the arrival time, the computation time, and the deadline; and the several other properties serve to describe the task and the scheduling environment. The operating system manages hardware resources, schedule those resources, and support the software architecture on which tasks are implemented. The hardware layer is beneath the operating system. In this thesis, we are mostly concerned with the nature of abstract timing requirements, the methods for scheduling analysis based on the task specification, and the mechanisms for scheduling the tasks in the operating system.

Two important properties of a task are the arrival characteristics and the nature of the deadline. The task arrivals may be periodic with a constant interval between successive invocations of the task, or the arrivals may be characterized by a statistical distribution where the inter-arrival time is a random variable. Tasks should complete execution before their deadlines. In some cases, a task absolutely must complete its computation by the specified deadlines; these are known as hard deadlines. If a task misses such a deadline, the value of completing the task after the deadline is nil. Missing hard deadlines might even result in the catastrophic failure of the system, depending on the task and the nature of its timing requirements. On the other hand, a task deadline may just specify a preference for completion time is called soft deadline, and there is still some value to the system in completing the task after the deadline has passed.

The notion of hard and soft real-time system requirements plays an important role in the design of the system software. If the system has hard real-time requirements, the designer must go to great lengths to guarantee that no deadlines will be missed (at least under expected operating conditions). If the system timing requirements are soft, the designer can choose a relaxed approach to system software.

1.2 Real Time Systems

Real-Time systems span several domains of computer science [20]. They are defense and space systems, networked multimedia systems, embedded automotive electronics etc. In a real-time system the correctness of the system behavior depends not only the logical results of the computations, but also on the physical instant at which these results are produced. A real-time system changes its state as a function of physical time, e.g., a chemical reaction continues to change its state even after its controlling computer system has stopped. Based on this a real-time system can be decomposed into a set of subsystems i.e., the controlled object, the real-time computer system and the human operator. A real-time computer system must react to stimuli from the controlled object (or the operator) within time intervals dictated by its environment. The instant at which a result is produced is called a deadline. If the result has utility even after the deadline has passed, the deadline is classified as soft, otherwise it is firm. If a catastrophe could result if a firm deadline is missed, the deadline is hard. Commands and Control systems, Air traffic control systems are examples for hard real-time systems. On-line transaction systems, airline reservation systems are soft real-time systems.

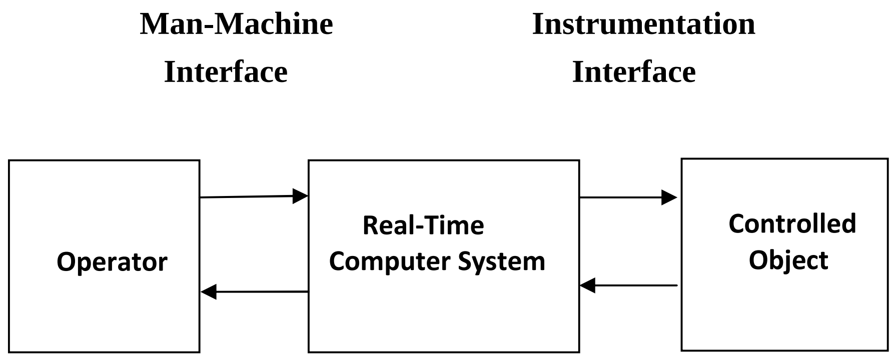


Figure 1.1: Real-Time System

Real-Time Systems [20] can be classified from different perspectives. The first two classifications, hard real-time versus soft real-time, and fail-safe versus fail-operational, depend on the characteristics of the application, i.e., on factors outside the computer system. The second three classifications, guaranteed-timeliness versus best-effort, resource-adequate versus resource-inadequate, and event-triggered versus time-triggered, depend on the design and implementation, i.e., on factors inside the computer system.

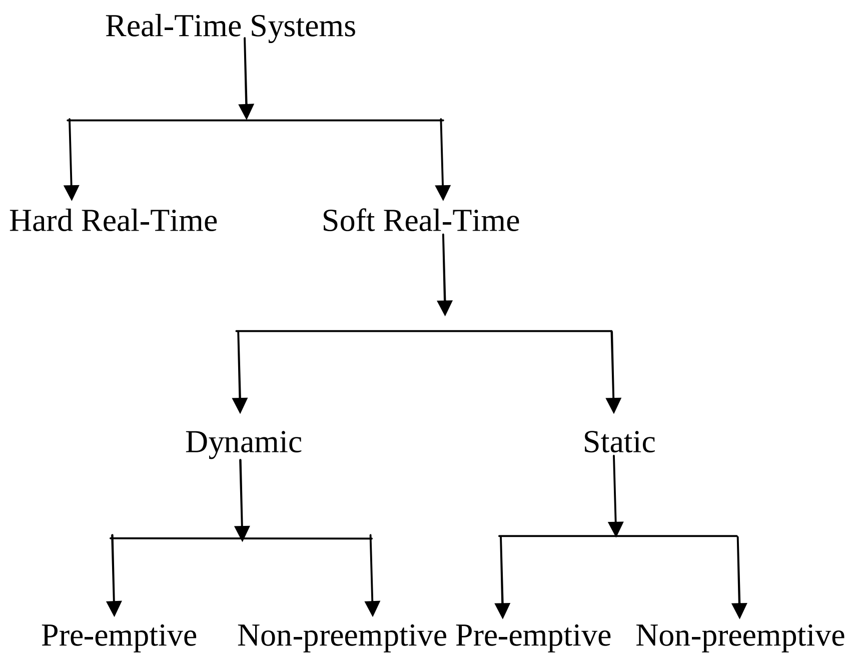


Figure 1.2: Classification of Real-Time Systems

1.3 Hard Real-Time and Soft Real-Time

The response time requirements of hard real-time systems are in the order of milliseconds or less and can result in a catastrophe if not met [5]. In contrast, the response time requirements of soft real-time systems are higher and not very stringent. In a hard real-time system, the peak-load performance must be predictable and should not violate the predefined deadlines. In a soft real-time system, a degraded operation in a rarely occurring peak load can be tolerated. A hard real-time system must remain synchronous with the state of the environment in all cases. On the other hand, soft real-time systems will slow down their response time if the load is very high. Hard real-time systems are often safety critical. Hard real-time systems have small data files and real-time databases. Temporal accuracy is often the concern here. Soft real-time systems for example, on-line reservation systems have larger databases and require long-term integrity of real-time systems. If an error occurs in a soft real-time system, the computation is rolled back to a previously established checkpoint to initiate a recovery action. In hard real-time systems, roll-back/recovery is of limited use.

Characteristic	**Hard real-time**	**Soft real-time**
Response Time	Hard-required	Soft-designed
Peak-load performance	Predictable	Degraded
Control of place	Environment	Computer
Safety	Often critical	Non-critical
Size of data files	Small/medium	Large
Redundancy Type	Active	Checkpoint-recovey
Data integrity	Short-term	Long-term
Error detection	Autonomous	User assisted

Table 1.1: Difference between Hard real-time and Soft real-time

1.4 Real-Time Scheduling

Scheduling involves the allocation of resources and time to tasks in such a way that certain performance requirements are met. A hard real-time system must execute a set of concurrent real-time tasks in such a way that all time-critical tasks meet their specified deadlines [20]. Every task needs computational and data resources to complete the job. The scheduling problem is concerned with the allocation of the resources to satisfy the timing constraints. Figure 1.2 given above represents taxonomy of real-time scheduling algorithms.

The present paper focuses on scheduling algorithms for soft real-time. Hard real-time scheduling can be broadly classifies into two types: static and dynamic. In static scheduling, the scheduling decisions are made at compile time. A run-time schedule is generated off-line based on the prior knowledge of task-set parameters, e.g., maximum execution times, precedence constraints, mutual exclusion constraints, and deadlines. So run-time overhead is small. More details on static scheduling can be found in [30]. On the other hand, dynamic scheduling makes its scheduling decisions at run time, selecting one out of the current set of ready tasks. Dynamic schedulers are flexible and adaptive. But they can incur significant overheads because of run-time processing. Preemptive or non-preemptive scheduling of tasks is possible with static and dynamic scheduling. In preemptive scheduling, the currently executing task will be preempted upon arrival of a higher priority task. In non-preemptive scheduling, the currently executing task will not be preempted until completion.

1.5 Real-Time Operating System

A Real-Time Operating System (RTOS) [3] is not simply a real-time system. It is the core part of any real-time system. A real-time system includes all the system elements such as hardware, middleware, applications, communications and I/O devices. All the elements are needed to meet the system requirements. However, RTOS provides sufficient functionality to enable a real-time application to meet its requirements. It is also important to distinguish between a fast operating system and a RTOS. Speed, although useful for meeting the overall requirements, by itself is not sufficient to determine whether a system meets the requirements for an RTOS.

1.5.1 POSIX 1003.1 for RTOS

The IEEE Computer Society’s Portable Application Standards Committee (PASC) defined a standard for Portable Operating System Interface (POSIX) [24, 26]. This IEEE Standard 1003.1 includes IEEE Standard 1003.1a, IEEE Standard 1003.1b, and 1003.1c, IEEE Standard 1003.1d/j/q, and IEEE Standard 1003.13. IEEE Standard 1003.1a is the base for all the POSIX standards. IEEE Standard 1003.1b (formerly POSIX 1003.4) defines the needed real-time extensions. IEEE Standard 1003.1c defines the functionality of threads. These various standards have been combined by the Austin Group in producing IEEE standard 1003.1-2001. The latest version is now known as the IEEE 1003.1 2004 Edition. POSIX 1003.1b provides the standard criteria for RTOS services and is designed to allow programmers to write applications that can easily be ported to any Operating Systems (OS) that is POSIX compliant. The basic RTOS services covered by POSIX 1003.1b include asynchronous I/O, synchronous I/O, memory locking, semaphores, shared memory, timers, Inter-Process Communication (IPC), real-time files, real-time threads, and scheduling. Real-time scheduling is the most important feature of a RTOS. POSIX 1003.1b specifies the following scheduling policies. FIFO - Priority based preemptive scheduling, FIFO is used among tasks with the same priority. RR - Processes with same priority use Round Robin policy. A process executes for a quantum of time; and then it is moved to the end of the queue corresponding to its priority level. Higher priority tasks can preempt lower tasks. The size of the quantum can be fixed, configurable, or specific for each priority level.

1.5.2 RTOS Examples

a. Microsoft Windows CE – Non-Linux Based Commercial RTOS
Microsoft Windows CE is designed as a general-purpose and portable real-time operating system for small memory, 32-bit mobile devices. Windows CE slices CPU time among threads and provides 256 priority levels. To optimize performance, all threads are enabled to run in kernel mode. All non-preemptive portions of the kernel are broken into small sections reducing the duration of non-preemptive code. Windows CE incurs long latencies for tasks.

b. VxWorks – Commercial RTOS
VxWorks, by Wind River Systems, is a real-time operating system. It runs currently on its own kernel. However, its development is done on a host machine such as Linux or Windows. Its cross-compiled target software can be run on various target CPU architectures. VxWorks runs in supervisor mode, and does not use traps for system calls. VxWorks supports priority interrupt-driven preemption and optional round-robin time slicing. The micro kernel supports 256 priority levels. VxWorks supports some of the IEEE POSIX 1003.1 functions.

c. LynxOS – POSIX Compatible Commercial RTOS

LynxOS is a POSIX compatible, multithreaded OS designed for complex real-time applications that require fast, deterministic response. It is scalable from small, embedded products to large switching systems. The micro-kernel can schedule, dispatch interrupts, and synchronize tasks. It uses scheduling policies such as prioritized FIFO, Dynamic Deadline Monotonic (DDM, the shorter the dynamic deadline, the higher is its priority) scheduling, time-slicing, etc. It has 512-priority levels and supports remote console and remote monitoring. For instance, LynxOS can be used as a hard real-time system for controlling gas levels in chemical plants remotely.

d. RTLinux – Open Source Linux-Based RTOS

RTLinux [25] is a hard real-time operating system that runs Linux as its lowest priority thread. The Linux thread is completely pre-emptible so that real-time threads and interrupt handlers are never delayed by non-real-time operations. Real-time applications can make use of all the powerful, non-real-time services of Linux. RTLinux scheduling policies supports EDF. RTLinux, originally developed at the New Mexico Institute of Technology, is an open-source product. RTLinux-specific components are released under the GNU General Public License (GPL), and Linux components are released under the standard Linux license. The source code is freely distributed. Non-GPL versions of the RTLinux components are available from FSMLabs [25].

e. RED-Linux – Open Source Linux-Based RTOS

RED-Linux [27] is an open-source real-time and embedded version of Linux version 2.2.14. In addition to the original Linux capability, it improves the real-time behaviors of the Linux kernel in many ways. RED-Linux supports a short kernel blocking time, a quick task response time and, modularized runtime General Scheduler Interface (GSI) so that different scheduling methods can be selected depending on the application.

f. KURT-Linux – Open Source Linux-Based RTOS

KU Real-Time Linux (KURT) [28] is a Linux system with real-time modifications to allows scheduling of real-time events at 10's of microseconds resolution. Rather than relying on priority based scheduling or strictly periodic schedules, KURT relies on application specified schedules.

KURT can function in two modes: focused mode, where only real-time processes are allowed to run; and mixed mode, where the execution of real-time processes still takes precedence, but non-real-time processes are allowed to run when real-time tasks are not running. KURT was developed by the Information and Telecommunication Technology Center (ITTC) at the University of Kansas. KURT may be used and distributed according to the terms of the GNU Public License.

1.6 Organization of the thesis

The remaining part of the thesis is organized as follows: in chapter two, real time system background, nature of the disk scheduling problem, problem definition and objectives of the dissertation is discussed. Chapter 3 describes the existing real-time disk scheduling algorithms are surveyed and what study has been made to get the dissertation done. Chapter 4 describes the evaluation model characterization and its assumptions. Each module of the evaluation model and its algorithm is described in chapter 5. Chapter 6 describes the implementation detail of the evaluation model. In chapter 7, the analysis and experimentation of the algorithm is described with the use of simulator. The results of the algorithms with respect to the different input set are also observed in this chapter. This chapter is the major part of this dissertation. Finally, the conclusion of the work and some possible future extensions are presented in chapter 8.

CHAPTER 2

BACKGROUND AND PROBLEM DEFINITION

2.1 Background

While processing speed in computer systems have been increased rapidly in last two decades, the improvement in access time to stable and reliable mass storage has not been as much as processing time. On average, processing speed tends to increase 55% per year, while disk access speed only increases about 7% [9]. This causes a large bottleneck in modern computers and tasks that have a specified deadline such as created ones in multimedia applications. On the other hand, one of the requirements of real-time applications is Quality of Service (QOS) guarantee by operating system .These applications are categorized based on the strictness of their QOS requirements as soft or hard real-time applications. In soft real-time applications such as video/audio playback, the most important QOS requirements are minimizing the number of requests that miss deadlines instead of maximizing the system throughput.
Recently, interest in real-time systems research has been growing. In a real-time system, each computational entity has a deadline when submitted to the system [7] .These entities must be scheduled and processed in such a way that they complete by their corresponding deadlines. Generally, two categories of real-time systems can be identified. In a hard real-time system, missing a deadline may lead to a catastrophe, while in a soft real-time system, missing a deadline may only reduce the ‘value’ of the entity to the system.

2.2 Nature of the disk scheduling problem

To service a disk request, several operations take place [22]. First, the disk head must be moved to the appropriate cylinder (seek time). Then, the portion of the disk on which the disk page is stored must be rotated until it is immediately under the disk head (latency time). Then, the disk page must be made to spin by the disk head (transmission time). The above components are needed to service a disk request are illustrated in figure 2.1.

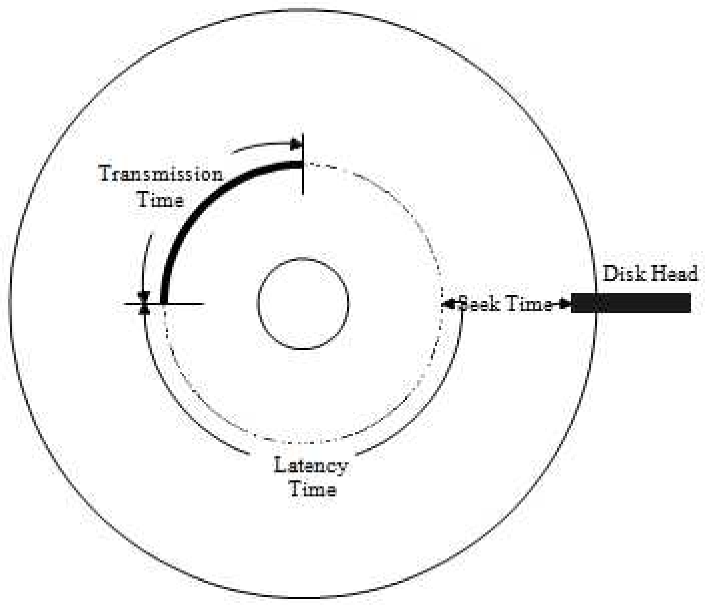


Figure 2.1: Components of a disk access

Queues build up for each disk because the inter-arrival time of the disk requests can be smaller than the time required by the disk to service a disk request. Disk Scheduling involves a careful examination of the pending disk requests to determine the most efficient way to service the disk requests. Disk scheduling problem involves reordering the disk requests in the disk queue so that the disk requests will be serviced with the minimum mechanical motion by employing seek optimization and latency optimization. The disk scheduling problem can be reduced to the travelling salesman problem which is a classical graph theory problem known to be NP-complete. Thus, the disk scheduling problem is an NP-complete problem [23] .The desirable characteristics of the disk scheduling algorithms are maximizing throughput, being fair, minimizing the mean response time, and minimizing the variance of the response times (predictability). Traditionally, disk scheduling algorithms have mainly been concerned with increasing the bandwidth utilization of the disks, by ordering disk requests so that the seek time is minimized. Heuristic algorithms such as Shortest Seek Time First (SSTF), SCAN and Circular SCAN addresses this problem. When real-time constraints are imposed on disk requests, minimizing seek time alone is not sufficient. Real-time algorithms should address both minimizing the seek time and satisfying the timing constrains. Existing real-time disk scheduling algorithms are discussed in the following section.

2.3 Problem Definition:

In a disk-based real time system [31], disk I/O occupies a major portion of transaction execution time. Disk scheduling involves a careful examination of the pending disk requests to determine the most efficient way to service the disk requests.
The disk scheduling problem involves reordering the disk requests in the disk queue so that the disk requests will be serviced with the minimum mechanical motion by employing seek optimization and latency optimization.

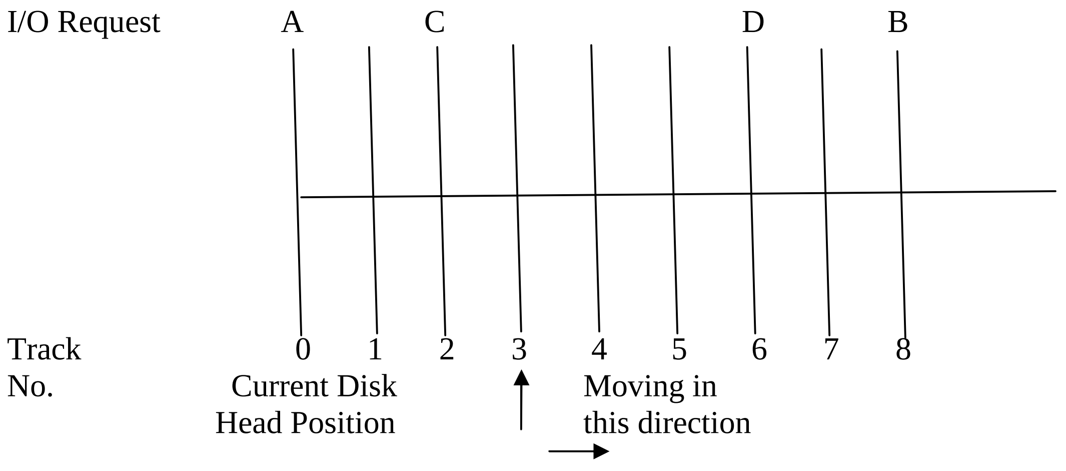


Figure 2.2: Disk Scheduling Problem

In a real time systems, when request may have to be satisfied within deadlines algorithm such as earliest deadline first(EDF) and least slack time first we used.

The earliest deadline first algorithm is shown to be optimal if the service times of the requests are known in advance [2]. However, the disk service time for a request depends on the relative position of the request from the current position of the read-write head. Recently it has been shown that even when the tasks are non-preemptable, EDF [1] is an optimal policy. However, due to the overheads of seek time, strict real time scheduling of a disk arm may result in excessive time of cost and poor utilization of the disk.

Traditionally, disks have used seek optimization techniques such as SCAN or SSTF for minimizing arm movement in serving request. These techniques reduce disk arm utilization by serving requests close to the disk arm. The request queue is ordered by the relative position of requests to the disk surface to reduce seek overheads. Even though these techniques utilize the disk arm efficiently, they may not be suitable for real time environments since they do not have a notion of time or deadline in making a scheduling decision.

Another requirement is that the scheduling algorithm should be fair. For example, SSTF is not a fair scheduling algorithm since requests at the edges of the disk surface may get starved. If the scheduling algorithm is not fair, occasional requests in the stream may get starved of service and hence will result in missed deadlines. Hence; we will not use SSTF type algorithms in this dissertation.

In this dissertation, a scheduling algorithm SCAN-EDF is proposed. SCAN-EDF is a hybrid algorithm that incorporates the real time aspects of EDF and seeks optimization aspects of SCAN, CSCAN and other such seek optimization policies. We will show that SCAN-EDF has good performance than EDF and CSCAN.

2.4 Objectives

The objectives of this research work are:

1. To design and evaluation model to simulate the performance analysis of different types of real time disk scheduling algorithm.
2. To characterize the SCAN-EDF, EDF and CSCAN real time disk scheduling algorithms under different statistics.

CHAPTER 3

RELATED WORK

3.1 Motivation

Since performance criteria for real-time systems are quite different from that of conventional systems, and since research on real-time systems is still its infancy, there are many new and challenging issues raised in designing such a system. One of these challenging issues is real-time I/O scheduling. Because I/O devices are orders of magnitude slower than CPU speeds, the improvement of I/O efficiency is extremely important to the performance of a real-time system. This motivates our interest in examining the real-time disk scheduling problem.

3.2 Literature Survey

In FCFS algorithm [12], the disk controller processes the I/O requests in the order in which they arrive, this policy aims to minimize response time with few regard to throughput. Therefore, FCFS decreases the response time while other traditional disk scheduling algorithms, such as SCAN, C-SCAN, LOOK and SSTF reduce average seeks and increases its throughput.

Group-Sweeping Scheduling (GSS) [13, 14] is another disk scheduling strategy in which requests are served in cycles with a round-robin manner.

To reduce disk arm movements, the set of streams is divided into some groups that are served in a fixed order and streams within a group are served according to SCAN. The mentioned algorithms do not consider real-time constraints of I/O tasks and therefore are not suitable to be applied directly on a real time system [15]. On the other hand, some other disk scheduling algorithms such as Earliest-deadline-first (EDF) [16] addresses this issue without considering disk-access time. EDF policy is optimal if the tasks are independent [2,4], therefore, it is not proper for real time disks tasks because in the disk scheduling problem, the service time of a task depends on the track location of the previous task [2]. Therefore, it seems to have a proper algorithm for real time disk scheduling, it is better to modify and combine some methods. Earliest-Deadline-SCAN (D-SCAN) [16] is a modification of the traditional SCAN algorithm to consider request deadlines. In D-SCAN, the track location of the requests with earliest deadline is used to determine the scan direction. In Feasible-Deadline-SCAN (FD-SCAN) algorithm [17], the track of the request with the feasible deadline is used to determine the scan direction.

As another combined method; SCAN-EDF is one of the well-known real-time disk scheduling algorithms [2, 18]. This method improves both disk seek-time and miss ratio and utilizes SCAN to reschedule tasks in a real-time EDF schedule. Since tasks rescheduled in SCAN-EDF should have the similar deadlines, its efficiency depends on the number of tasks with the similar deadlines. If all the tasks have not near deadlines, the schedule result of SCAN-EDF would be the same as EDF. In SCAN-EDF algorithm, rescheduling is only possible within a local group of requests. To overcome this problem, Deadline-Modification-SCAN (DM-SCAN) [29] suggests the use of Scannable-groups. In this algorithm, request deadlines are reduced several times during the process of rescheduling to pre-serve EDF schedule. Unlike DM-SCAN, Reschedulable-group-SCAN (RG-SCAN) [29] does not require its input disk requests to be sorted by their deadlines. It also forms larger groups without any deadline modification.

In SCAN-EDF, DM-SCAN and RG-SCAN algorithms rescheduling is only possible within a local group of requests. [29] Suggests Global Seek-optimization Real-Time (GSR) disk scheduling algorithm that groups the EDF input tasks based on their scan direction. These tasks are moved to their suitable groups to improve the system performance in terms of increased throughput and decreased number of requests that miss deadlines. GSR schedules are always feasible if the input real-time requests are EDF feasible, but they may be infeasible if the input schedule is infeasible. Also, this method can be applied on periodic and aperiodic real time requests.

3.3 EDF Scheduling

The EDF scheduling algorithm is a priority driven algorithm in which higher priority is assigned to the request that has earlier deadline, and a higher priority request always preempts the lower priority one. This scheduling algorithm is an example of priority driven algorithms with dynamic priority assignment in the sense that the priority of a request is assigned as the request arrives. EDF is also called deadline-monotonic scheduling algorithm. The earliest deadline first (EDF) algorithm is optimal if requests service times are known in advance [2]. However, the disk-service time for a request depends on its position relative to the read-write head's current position. The original EDF algorithm assumed that tasks were pre-emptable with zero preemption cost and showed that EDF could schedule tasks if and only if the task utilization were less than 1. However, current disks are not pre-emptable. EDF gives each real-time request a deadline and serves requests strictly in order. Strict real-time scheduling of the disk-arm might result in excessive seek-time cost and poor disk utilization.

While EDF schedules real-time requests, the immediate server approach serves aperiodic requests, giving them priority for service immediately after the current real-time request. This schedule allows a certain number of aperiodic requests during each round of service. When there are only a few aperiodic requests, the real time requests use the remaining service time during that round. This policy, which provides reasonable response times for aperiodic requests while guaranteeing deadlines for real-time requests, contrasts with earlier approaches that, guaranteed timely service only for real-time requests.

3.4 CSCAN

CScan is a Scan type of disk-scheduling algorithm [1]. The disk read-write head scans for request in one direction, from the outermost to the innermost track or vice-versa, serving requests in the order of its scan direction. If it is scanning inward, after it servers the innermost request the head moves to the outermost pending request. This policy does seek optimization while guaranteeing that no request gets starved of service. Since it has no notion of time or deadlines, it serves real-time requests strictly in the order of their location on the disk surface. It also serves aperiodic requests in scan order, and they might have to wait behind a number of real-time requests.

3.5 SCAN-EDF

The SCAN-EDF disk scheduling algorithm [1] combines seek optimization techniques and EDF in the following way: Requests with earliest deadline are served first. But, if several requests have the same deadline, these requests are served by their track locations on the disk or by using a seek optimization scheduling algorithm for these requests. This strategy combines the benefits of both real-time and seeks optimization scheduling algorithms. Requests with earlier deadlines are served first, but requests with the same deadline make use of seek optimization techniques to reduce disk utilization [1].SCAN-EDF applies seek optimization to only those requests having the same deadline. A more precise description of the algorithm is given below:

SCAN-EDF algorithm

Step 1: Let T=set of requests with the earliest deadline

Step 2: if |T|=1, (there is only a single request in T), service the request.

else let t1 be the first task in T in scan direction, service t1.

Go to **Step 1.**

The scan direction can be chosen in several ways. In Step 2, if the tasks are ordered with the track numbers of tasks such that N1<=N2<=…<=N1, then we obtain a CSCAN type of scheduling where the scan takes place only from smallest track number to the largest track number. If the tasks are ordered such that N1>=N2>=…=N1, then we obtain CSCAN type of scheduling where the scan takes place only from the largest track number to the smallest track number. If the tasks can be ordered in either of the above forms depending on the relative position of the disk arm, we get SCAN type of algorithm.

SCAN-EDF can be implemented with a slight modification to EDF. Let D_i be the deadlines of the tasks and N_i be their track positions. Then, the deadlines can be modified to be $D_i+f\ (N_i)$, where f () is a function that converts the track numbers of the tasks into small perturbations to the deadlines. The perturbations have to be small enough such that $D_i + f\ (N_i) > D_j + f\ (N_j)$, *if* $D_i > D_j$ and requests i and j are ordered in the SCAN order when $D_i= D_j$. We can choose f () as f $(N_i) = N_i/N_{max}-1$, where N_{max} is the maximum track number on the disk or some other suitably large constant. For example, let tasks A, B, C and D have deadlines 500,500,500 and 600 respectively and ask for data from tracks 347, 113, 851, and 256 respectively. If N_{max}=1000, the modified deadlines of A, B, C and D become 499.347, 499.113, 499.851 and 599.256 respectively when we use f $(N_i) = N_i/N_{max}-1$. When these requests are served by their modified deadlines, requests A, B and C is served in the SCAN order of B, A and C and request D is served later.

3.6 Parameters

There are various algorithms available for the real time disk scheduling work. We need to know how good they are. That is some criteria are needed to evaluate the performance of those algorithms [31].

3.6.1 Transaction Request Parameters

3.6.1.1 Arrival Time (AT) or Release Time

The time at which request for the transaction on the disk arrives is the Arrival time for that request.

Inter arrival time: Arrival Time difference of two consecutive requests.

Inter arrival time= (1/Transaction arrival rate) (log 1/Block accessed)

Arrival Time=Arrival Time + Inter arrival Time.

3.6.1.2 Block Size

It may be called as request size. It is the number of block to be accessed from the disk or we may say number of block residing on disk on which transaction operation (Read, Write, Read-Write or Read-write-compute) has to be performed.

3.6.1.3 Block Access

It may be called as block location on disk. It is analogous to the track location or sector location on the disk.

3.6.2 Scheduling Parameters

3.6.2.1 Transaction ID

Whenever any request for the transaction on the disk arrives, it is assigned an ID by which it can be identified by the scheduler. It is unique for every transaction that is arriving on the disk.

3.6.2.2 Average Execution Time (AET)

Average execution Time=1.5*Block Size

3.6.2.3 Deadline

The latest time at which a request must be completed is the deadline.

Deadline=Arrival Time+ (Slack factor*Average Execution Time)

Or

Deadline=Arrival Time + (Slack Factor * (1.5 * Block Size)

3.6.3 Disk parameters

3.6.3.1 Disk Size

Disk consists of number of tracks and number of tracks forms a sector and numbers of sectors form a cylinder.

3.6.3.2 Seek Time

Time required to move the disk head from the current position to the target track.

Seek Time= Seek factor*[abs (Block Access-Current Head Position)].

3.6.3.3 Rotational Latency:

Once the desired track has been reached, then time required to rotate the disk until desired sector is under the disk head.

3.6.3.4 Transfer Time

The time required for actual transfer of data between the disk and main memory.

Transfer Time=Block size*Transmission Factor.

CHAPTER 4

SPECIFICATION

In this 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 Disk Scheduling algorithms. This Evaluation Model is not machine oriented, means, the simulator is not designed following the core ideas of the 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, a brief specification of Evaluation Model is featured. In section 4.1, Evaluation Model is characterized with the machine specification, algorithm it runs and its working nature. It has some limited working criteria as assumed in section 4.2.

4.1 Evaluation Model Characterization

The Evaluation Model that is developed works for Real-Time Disk Scheduling algorithms. The Evaluation Model is a multi-scheduler simulator that is the consequence of the 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 Earliest Deadline First (EDF), SCAN-Earliest Deadline First (SCAN-EDF) and Circular-SCAN (C-SCAN). The applet generates the statistics of the scheduling algorithms for particular task. The task is a predefined set of tasks, with respective information about the tasks, which is used for the implementation of a scheduler.

4.2 Assumptions

The Evaluation Model has a limited exposure to the working of the disk 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 tasks have the following condition.

1. Each periodic task completes within its period.
2. No task is dependent on any other tasks.
4. Any non-periodic tasks have no deadlines.
5. Task preemption occurs instantaneously and with no overhead.
6. No consideration to memory or input/output has been made.

CHAPTER 5

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 described in chapter 4. 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 Disk 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, Swing, and Abstract Window Tool (AWT)**.**

5.1 Modules of Evaluation Model

The modules that are described below in this section have a meaning of logical entities that design the simulator. Each module has its own function; in each module the concept seem only logical. Different modules have the different types of function like accepting input for the scheduling algorithm, making computation, displaying output.
Later on, each of the below module is coded and integrated as a package. Each below modules that construct the simulator.

5.1.1 TrackCanavas

The trackcanavas module carries out the painting function to draw the track traversed by the jobs. It uses the abstract window tool (AWT) provided by the java programming language. It uses the graphics context, which demonstrates the graphics environment the applet is running for this.

Function paint is used for drawing the timeline for the particular scheduling algorithm. It takes the parameter as object Graphics and draws the timeline in the applet.

Algorithm for TrackCanvas Module

```
Step 1: Import attributes and methods from Canvas and class.
Step 2: If (Disk Scheduling algorithm)
        Import the attributes for paint () method from graphics
        Draw the resulting vector (tracks) according to the disk head travel
Step 3: End of algorithm.
```

Listing 5.1: Algorithm for TrackCanvas Module

5.1.2 DiskApplet

This module is responsible for taking the inputs of the scheduling. The input parameters such as track number and deadline are set by this module.

Algorithm for DiskApplet Module

```
Step 1: Setup of Layout manager for user interface.
Step 2: Setup panel (i.e. Graphical, Input, Manual output, Run, Title).
Step 3: Find out event occurs and user interface.
Step 4: Find out the type of Disk scheduling algorithm that user selected.
Step 5: According to the type of disk scheduling algorithm calculate that algorithm (i.e. call DScheduler).
Step 6: Setup display output and report according to disk scheduling algorithm (i.e. call TrackCanvas).
Step 7: End of algorithm.
```

Listing 5.2: Algorithm for DiskApplet module

5.2 Algorithm Configurations

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 the other 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.

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

5.2.1 Earliest Deadline First

Algorithm:

Step 1: Input track numbers and their respective deadlines.

Step 2: Sorted the track numbers according to deadlines (i.e. least deadline have first)

Step 3: calculate the tracks traversed, average seek length.

Step 4: End of algorithm.

Listing 5.3: Algorithm for Earliest Deadline First

5.2.2 SCAN- EDF

Algorithm:

Step 1: Input track numbers and their respective deadlines.

Sep2: Calculates the perturbation of the deadlines.

$D_i=D_i+N_i/1000$(Max track no.)

Step 3: sorted the track no. according to the perturbation of deadlines (i.e. least deadline have first)

Step 4: Calculate tracks traversed, average seek length.

Step 5: End of algorithm.

Listing 5.4: Algorithm for SCAN-EDF

5.3 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.

Algorithm:

Step 1: Import the attributes and functions of vector class.

Step 2: tokenizes the string input (i.e. input the track number to the vector data structure).

Step 3: According to type of disk scheduling algorithm, perform the disk scheduling algorithm in input vector.

Step 4: Result of disk scheduling algorithms is put into the output vector.

Step 5: Calculate the seek length of disk head movement, average seek length, traversed order by head according to disk scheduling algorithm.

Step 6: End of algorithm.

Listing 5.5: Algorithm for Scheduler

5.4 Integration Representation of Modules

The figure 5.1 shows the evaluation model for the construction of the simulator. The modules that are described in above section are integrated to corporate as a whole unit. Input set are defined with request with their respective deadlines. The simulator synthesizes the effect of the modules according to the input set for particular scheduling algorithms.

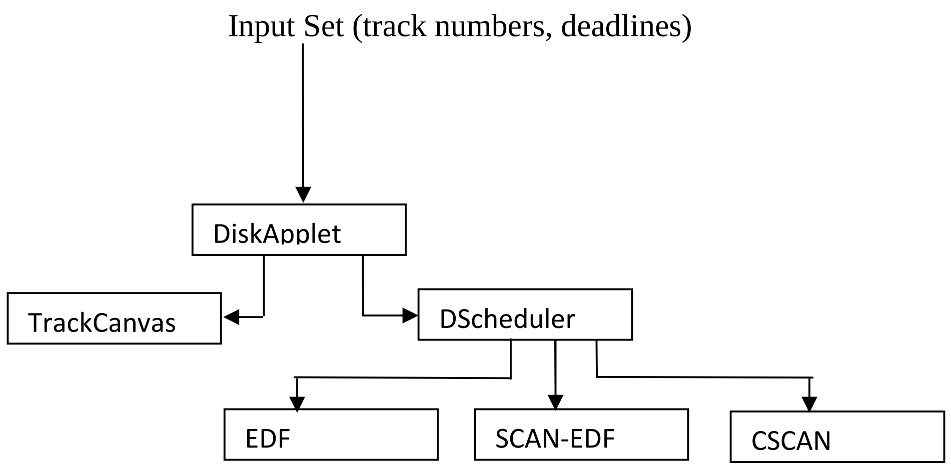


Figure 5.1: Pictorial Representation for the modules of a simulator

CHAPTER 6

IMPLEMENTATION

This chapter describes the implementation detail of the Evaluation Model. Since the dissertation gives the analytical result of the scheduling algorithms, this chapter shows the implementation sketch of the simulator. The simulator is used for making the analytical evaluation of the scheduling algorithms.

6.1 Implementation Tools

These are constructs that are used for the implementation of the Evaluation Model. In this dissertation, the simulator is a java applet which are tools for java programs that provides the effective generation of the applet. Java program files are carried out by these tools.

6.1.1 Javac

Javac is a java compiler which is used to compile all the programs implemented to build a simulator. The compiler creates class files (.class files) of all the 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 machine level instructions. The output of the javac code is directly executed by the Java Virtual Machine (JVM).

6.1.2 Data Structures

The data structures which are 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 vector is used to store the input task set.

6.2 Evaluation Model Development

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 modular way. This section of the dissertation describes the logical overview of the development model; the evaluation model is implemented by using the approach given below:

6.2.1 Phase 1

Input Determination

Input determination ensures the validation and completeness of the input parameters that are passed to the simulator. Input set, which is input to the scheduler are their track numbers and their respective deadlines.

6.2.2 Phase 2

Algorithm Execution

In this phase, execution of the scheduling algorithm is done. The input sets are scheduled according to the scheduling algorithm, with respect to the track number and deadlines. This phase sets the characteristics of scheduling algorithm to be executed. This is important phase for developing the simulator since the scheduling algorithms that are being used as part of the analysis needs to be executed in efficient and precise way.

i) Earliest Deadline First

The earliest deadline first (EDF) algorithm is optima1 if requests' service times are known in advance [2]. However, the disk-service time for a request depends on its position relative to the read-write head's current position. The original EDF algorithm assumed that tasks were preemptable with zero preemption cost and showed that EDF could schedule tasks if and only if the task utilization were less than 1. However, current disks are not pre-emptable. EDF gives each real-time request a deadline and serves requests strictly in order. Strict real-time scheduling of the disk arm might result in excessive seek-time cost and poor disk utilization. While EDF schedules real-time requests, the immediate server approach [5] serves aperiodic requests, giving them priority for service immediately after the current real-time request. This schedule allows a certain number of aperiodic requests during each round of service. When there are only a few aperiodic requests, the real time requests use the remaining service time during that round. This policy, which provides reasonable response times for aperiodic requests while guaranteeing deadlines for real-time requests, contrasts with earlier approaches that, guaranteed timely service only for real-time requests.

II) SCAN-EDF

SCAN-EDF is a hybrid scheduling algorithm that provides both seek optimization and earliest deadline first service [2]. Requests are normally served in EDF order. If several requests have the same deadline, they are served according to their track locations on the disk. Because Scan-EDF applies seek optimization only to requests having the same deadline, its efficiency depends on how often seek optimization can be applied. We can improve its efficiency with techniques that give various requests the same deadlines. Scan-EDF prescribes that requests have release times that are multiples of the period p. Hence, requests are grouped in batches and served accordingly. When the requests have different data-rate requirements, Scan-EDF can be combined with a periodic fill policy to give all the requests the same deadline. Requests are served in a cycle, and each request gets service time proportional to its required data rate. The length of the cycle is the sum of the service times of all requests. All requests in the current cycle are given a deadline at the end of the current cycle.

III) C-SCAN

CSCAN (for circular Scan) is a Scan type of disk-scheduling algorithm [2]. The disk read-write head scans for requests in one direction, from the outermost to the innermost track or vice versa, serving requests in the order of its scan direction. If it is scanning inward, after it serves the innermost request the head moves to the outermost pending request. This policy does seek optimization while guaranteeing that no request gets starved of service. Since it has no notion of time or deadlines, it serves real-time requests strictly in the order of their location on the disk surface. It also serves aperiodic requests in scan order, and they might have to wait behind a number of real-time requests.

6.2.3 Phase 3

Status Tracing

This phase gives the exact behavior of the scheduler when executing the particular scheduling algorithm. This phase gives the precise and clear information about the input set execution. This phase displays the output statistics like track traversed, track to be accessed, track traversed order, average seek length is computed for each input set for the EDF, SCAN-EDF and CSCAN algorithms.

6.2.4 Phase 4

Track Traversed

This phase animates the drawing of the execution of the input set as a track traversed. It gives the pictorial description of the execution of the each input set. The main aim of this phase is to provide ease the user interface and understanding the different types of scheduling behavior.

6.3 Inputs and Outputs

In this section, the layouts of the inputs and outputs of the Evaluation Model for particular scheduling algorithm are described. The algorithms have a specific format of their input and output. Each scheduling algorithms take input as input set with track number with respect to its deadline. The scheduler then schedules these tasks according to the algorithm. The outputs of the Evaluation Model are statistics and status of the algorithm like track traversed, track to be accessed, track traversed order, and average seek length.

6.3.1 Earliest Deadline First Algorithm

Input:

Input Set: Track numbers, Deadline

Evaluation:

Earliest deadline First Scheduler:

Execution of the algorithm

Output:

Statistics:

Tracks to be accessed, Total # Tracks Traversed, Tracks Traversed Order, Average Seek Length.

Status:

Algorithm traces step-by-step track traversed

6.3.2 SCAN-EDF

Input:

Input Set: Track numbers, Deadline

Evaluation:

SCAN-Earliest deadline First Scheduler:

Execution of the algorithm

Output:

Statistics:

Tracks to be accessed, Total # Tracks Traversed, Tracks Traversed Order, Average Seek Length.

Status:

Algorithm traces step-by-step track traversed

CHAPTER 7

ANALYSIS AND EXPERIMENTATION

This chapter explains the way analysis has been performed for the evaluation of EDF, SCAN-EDF and CSCAN algorithms. The analysis is carried out with the help of the simulator designed in the Java programming language. The working of the simulator is already explained in the previous chapters. In section 7.1, the description about the input set for the scheduling algorithms is given. The input sets are defined manually and experimental analysis is done for each input set. Description about given input statistics and computed output statistics with respect to the given input set is shown for each scheduling algorithms.

7.1 Input Set Analysis

Input sets are named separately, each with different number of input parameters. The result of the output statistics is observed. Track traversed, track to be accessed, track traversed order, average seek length, percentage of task completion within deadline is computed for each input set for the EDF, SCAN-EDF and CSCAN algorithms. This analysis reveals the description of the scheduling phenomenon occurring for a particular input set.

Input Set 1:

Track #	120	325	200
Deadline	5	7	4

Table 7.1.1 : Analysis of EDF , SCAN-EDF and CSCAN for input set 1.

Algorithm used: EDF

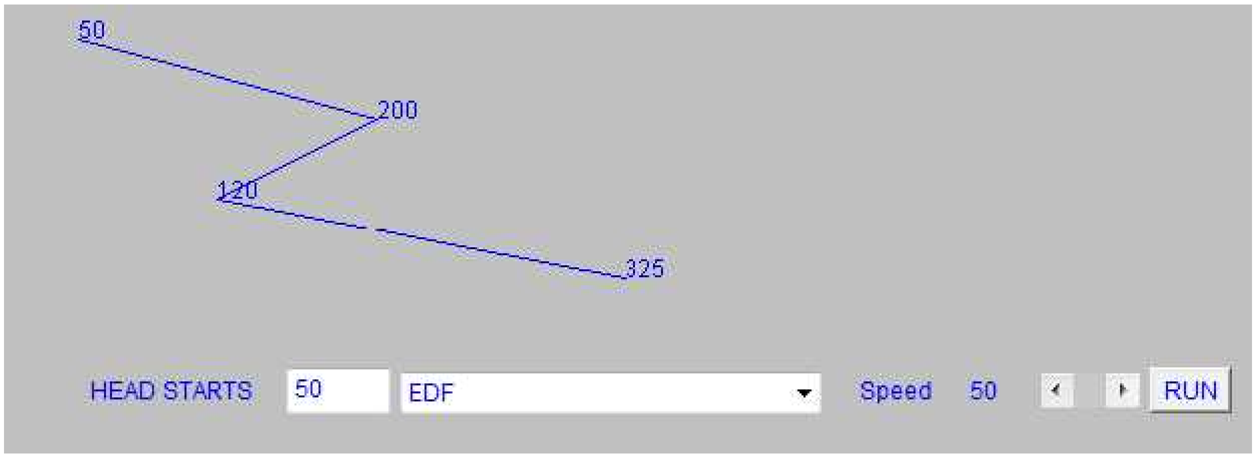
Tracks to be accessed: 3

Total # Tracks Traversed: 435

Tracks were Traversed in the following Order: 50 200 120 325

Tracks that are not completed tasks:

Average Seek Length: 145.0



Speed 50

HEAD STARTS 50

Figure 7.1.1: Execution of EDF for input set 1.

Algorithm used: SEDF

Tracks to be accessed: 3

Total # Tracks Traversed: 435

Tracks were Traversed in the following Order: 50 200 120 325

Average Seek Length: 145.0

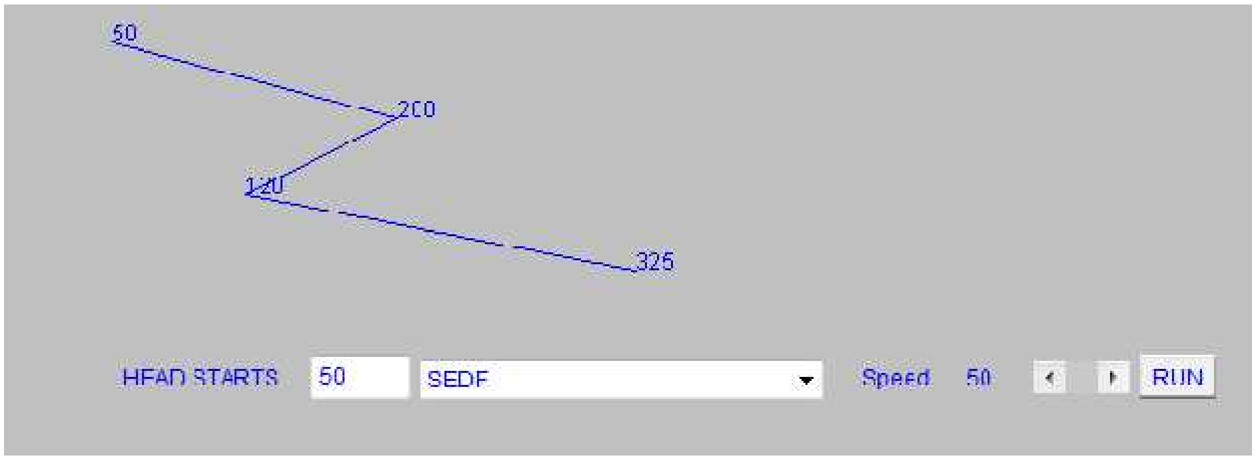


Figure 7.1.2: Execution of EDF for input set 1.

Algorithm used: CSCAN

Tracks to be accessed: 3

Total # Tracks Traversed: 449

Tracks were Traversed in the following Order: 50 120 200 325 499 0

Average Seek Length: 149.66667

Input set 2:

Track#	150	46	360	60	100
Deadline	3	1	6	3	5

Table 7.2.1: Analysis of EDF , SCAN-EDF and CSCAN for input set 2.

Algorithm used: EDF

Tracks to be accessed: 5

Total # Tracks Traversed: 498

Tracks were Traversed in the following Order: 50 46 150 60 100 360

Tracks that are not completed tasks:

Average Seek Length: 99.6

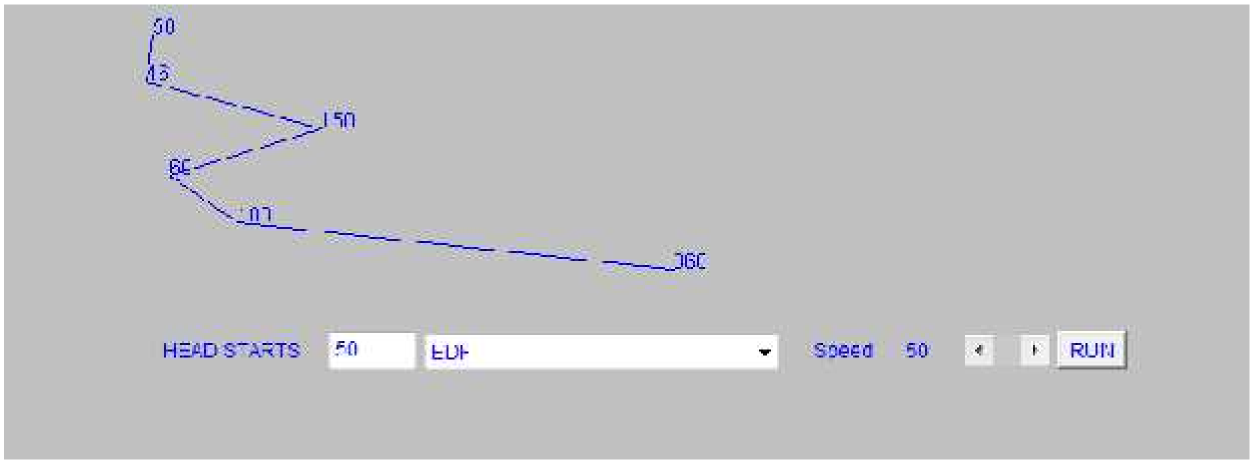


Figure 7.2.1: Execution of EDF for input set 2.

Algorithm used: SEDF

Tracks to be accessed: 5

Total # Tracks Traversed: 418

Tracks were Traversed in the following Order: 50 46 60 150 100 360

Average Seek Length: 83.6

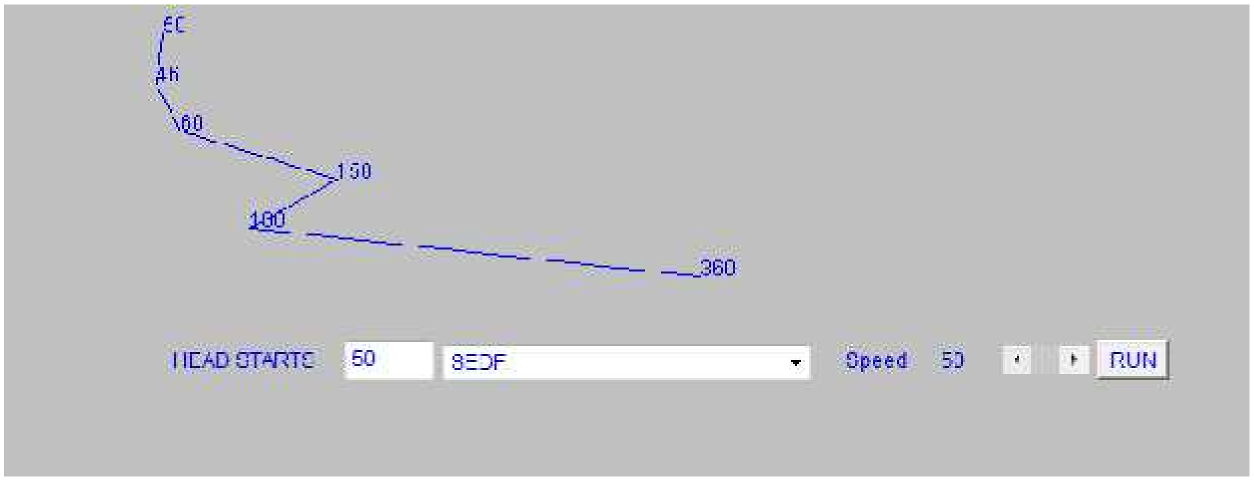


Figure 7.2.2: Execution of SCAN- EDF for input set 2.

Algorithm used: CSCAN

Tracks to be accessed: 5

Total # Tracks Traversed: 495

Tracks were Traversed in the following Order: 50 60 100 150 360 499 0 46

Average Seek Length: 99.0

Input set 3:

Track#	200	171	47	126	4	124	320	76
Deadline	4	6	2	7	6	4	5	5

Table 7.3.1: Analysis of EDF, SCAN-EDF and CSCAN for input set 3.

Algorithm used: EDF

Tracks to be accessed: 8

Total # Tracks Traversed: 1056

Tracks were Traversed in the following Order: 50 47 200 124 320 76 171 4 126

Tracks that are not completed tasks: 4

Average Seek Length: 132.0

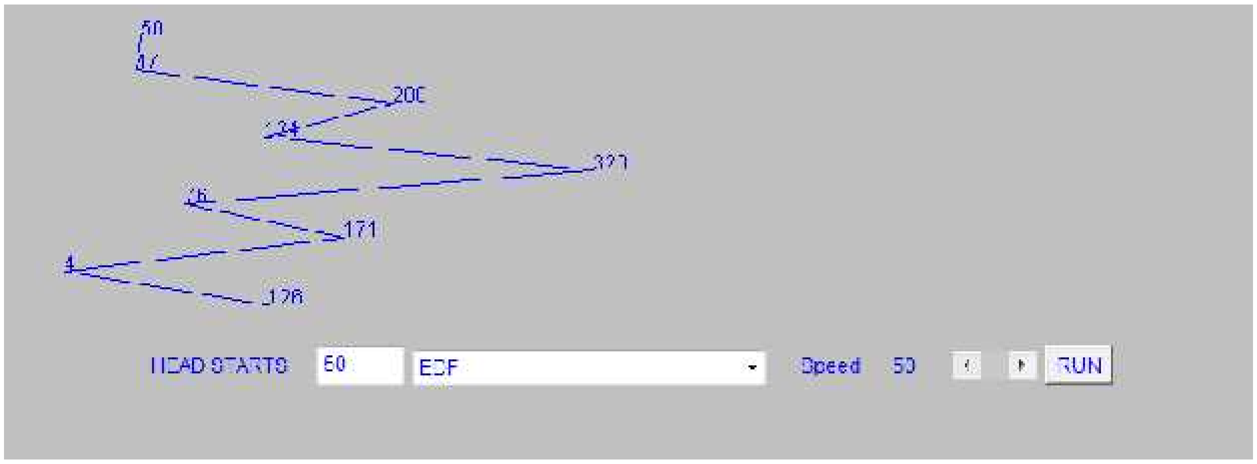


Figure 7.3.1: Execution of EDF for input set 3.

Algorithm used: SEDF

Tracks to be accessed: 8

Total # Tracks Traversed: 1052
Tracks were Traversed in the following Order: 50 47 124 200 76 320 4 171 126

Tracks that are not completed tasks: 171

Average Seek Length: 131.5

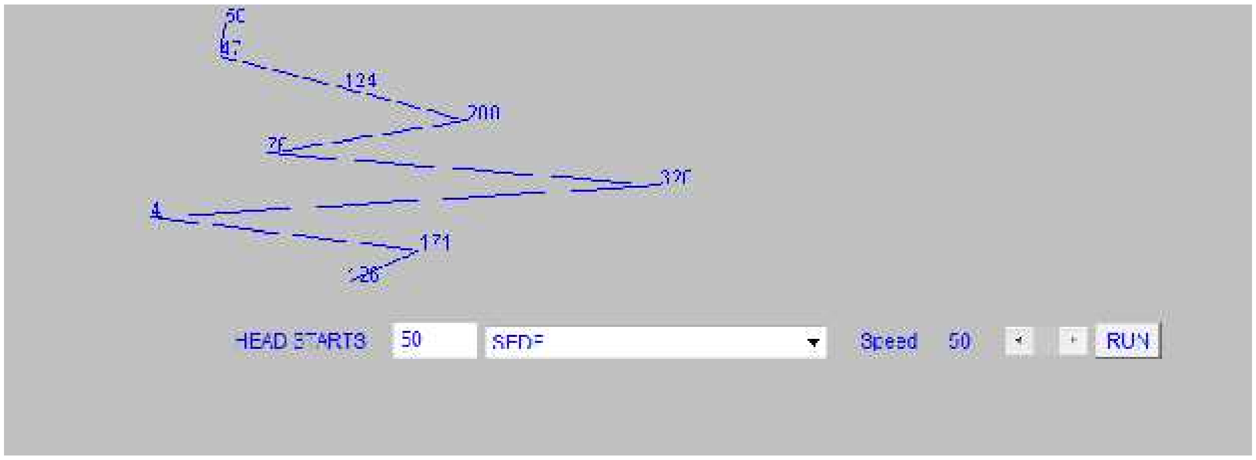


Figure 7.3.2: Execution of SCAN- EDF for input set 3.

Algorithm used: CSCAN

Tracks to be accessed: 8

Total # Tracks Traversed: 496

Tracks were Traversed in the following Order: 50 76 124 126 171 200 320 499 0 4 47

Average Seek Length: 62.0

Input set 4:

Track#	300	250	67	200	375	425	4	168	400	198
Deadline	4	5	8	4	6	8	3	6	4	5

Table 7.4.1: Analysis of EDF, SCAN-EDF and CSCAN for input set 4.

Algorithm used: EDF

Tracks to be accessed: 10

Total # Tracks Traversed: 1687

Tracks were Traversed in the following Order: 50 4 300 200 400 250 198 375 168 67 425

Tracks that are not completed tasks: 198   168

Average Seek Length: 168.7

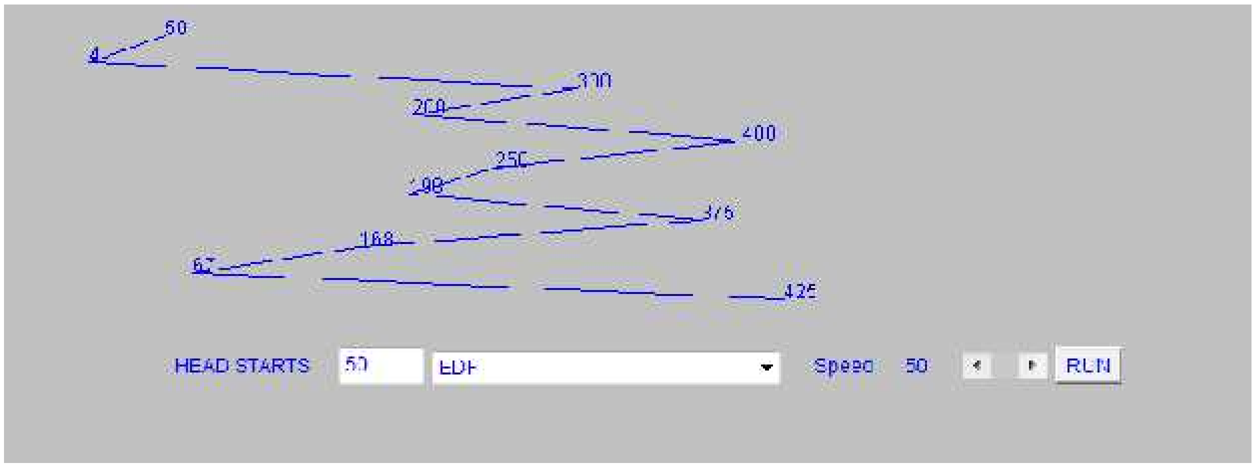


Figure 7.4.1: Execution of EDF for input set 4.

Algorithm used: SEDF

Tracks to be accessed: 10

Total # Tracks Traversed: 1651

Tracks were Traversed in the following Order: 50 4 200 300 400 198 250 168 375 67 425

Tracks that are not completed tasks: 250 375

Average Seek Length: 165.1

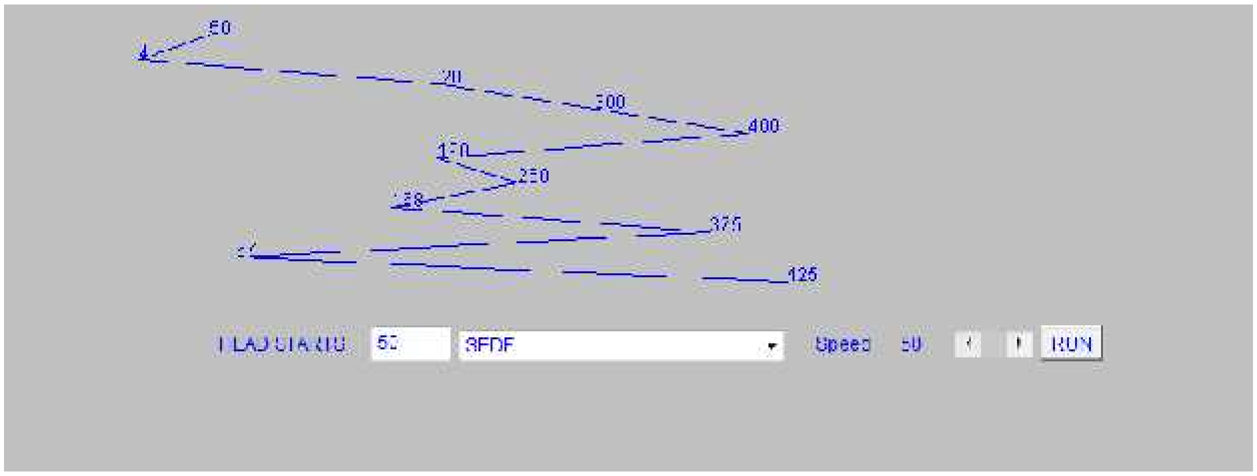


Figure 7.4.2: Execution of SCAN- EDF for input set 4.

Algorithm used: CSCAN

Tracks to be accessed: 10

Total # Tracks Traversed: 453

Tracks were Traversed in the following Order: 50 67 168 198 200 250 300 375 400 425 499 0 4

Average Seek Length: 45.3

Input set 5:

Track#	56	375	245	350	156	230	178	98	300	216	9	152
Deadline	4	4	3	7	3	7	7	9	9	3	4	3

Table 7.5.1: Analysis of EDF, SCAN-EDF and CSCAN for input set 5.

Algorithm used: EDF

Tracks to be accessed: 12

Total # Tracks Traversed: 2184

Tracks were Traversed in the following Order: 50 245 56 216 152 56 375 9 350 230 178 98 300

Tracks that are not completed tasks: 152 375 9

Average Seek Length: 165.3333

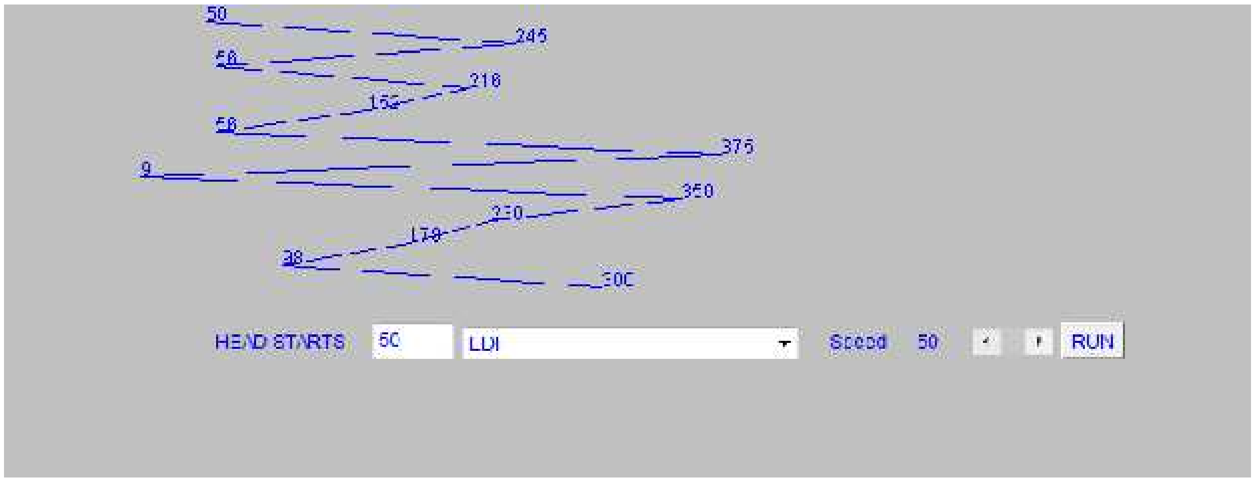


Figure 7.5.1: Execution of EDF for input set 5.

Algorithm used: SEDF

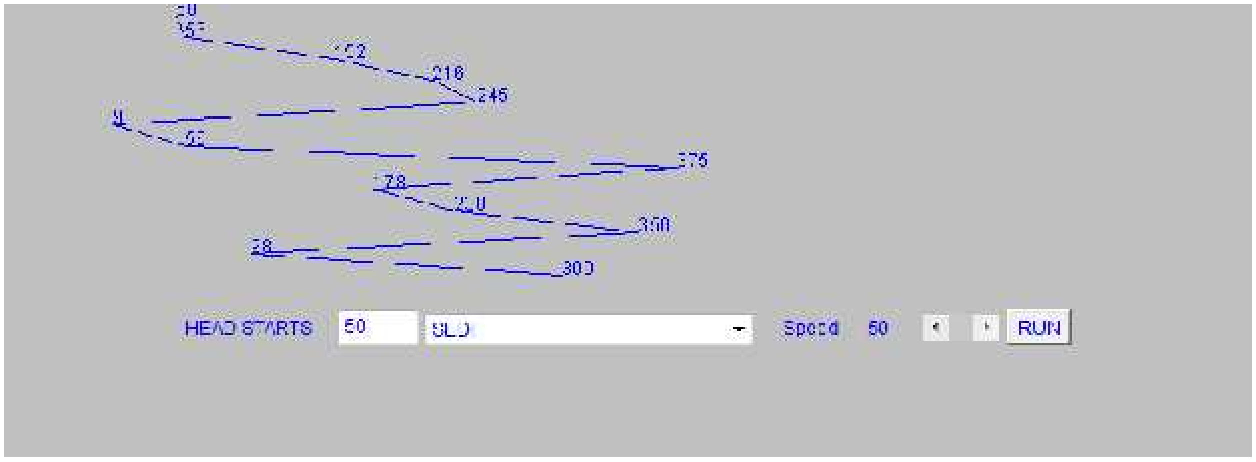
Tracks to be accessed: 12

Total # Tracks Traversed: 1620

Tracks were Traversed in the following Order: 50 56 152 216 245 9 56 375 178 230 350 98 300

Tracks that are not completed tasks: 245 56 375

Average Seek Length: 135.0



HEAD STARTS 50 SLD Speed 50 RUN

Figure 7.5.2: Execution of SCAN-EDF for input set 5

Algorithm used: CSCAN

Tracks to be accessed: 12

Total # Tracks Traversed: 505

Tracks were Traversed in the following Order: 50 56 98 152 178 216 230 245 300 350 375 499 0 9 56

Average Seek Length: 38.166668

Input set 6:

Track#	245	56	225	13	256	356	28	124	276	325	476	34	312	54	350
Deadline	8	3	8	3	9	6	9	7	9	5	3	8	8	7	3

Table 7.6.1: Analysis of EDF, SCAN-EDF and CSCAN for input set 6.

Algorithm used: EDF

Tracks to be accessed: 15

Total # Tracks Traversed: 2208

Tracks were Traversed in the following Order: 50 56 13 476 350 325 356 124 54 245 225 34 312 256 28 276

Tracks that are not completed tasks: 350 225 34 312 28 276

Average Seek Length: 147.2

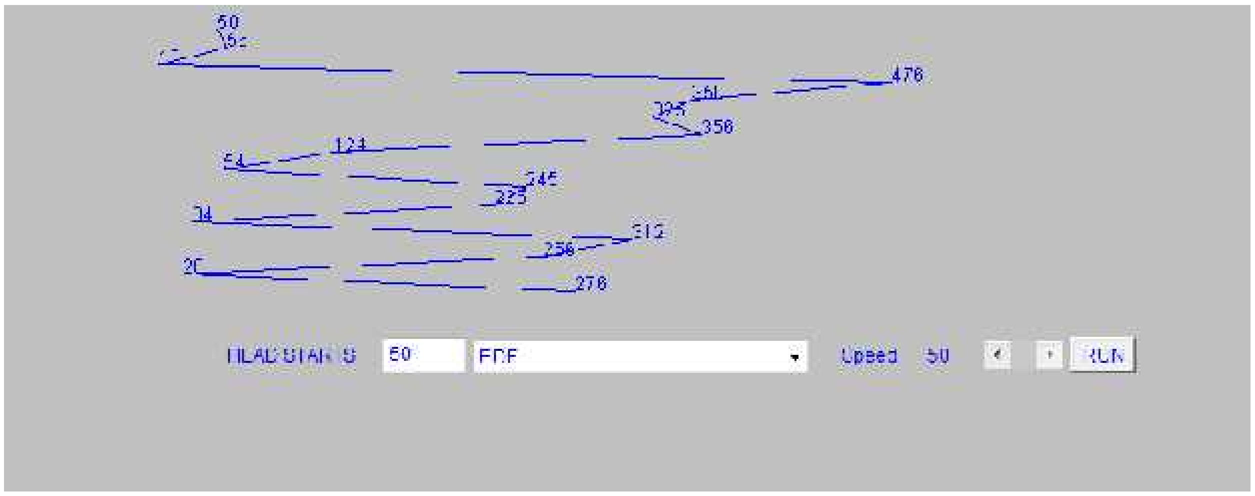


Figure 7.6.1: Execution of EDF for input set 6

Algorithm used: SEDF

Tracks to be accessed: 15

Total # Tracks Traversed: 1954

Tracks were Traversed in the following Order: 50 13 56 350 476 325 356 54 124 34 225 245 312 28 256 276

Tracks that are not completed tasks: 476 225 245 312 256 276

Average Seek Length: 130.26666

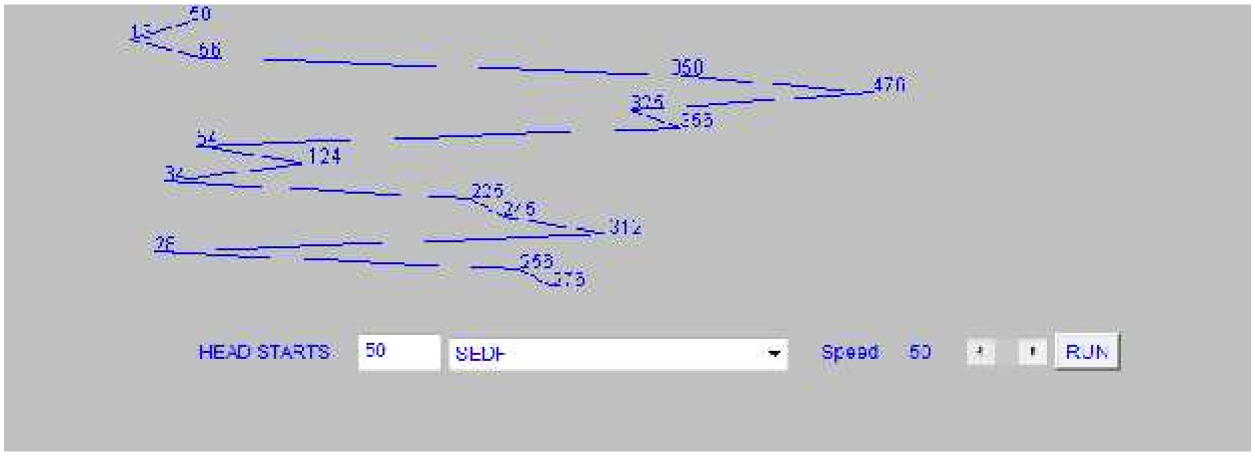


Figure 7.6.2: Execution of SCAN-EDF for input set 6

Algorithm used: CSCAN

Tracks to be accessed: 15

Total # Tracks Traversed: 483

Tracks were Traversed in the following Order: 50 54 56 124 225 245 256 276 312 325 350 356 476 499 0 13 28 34

Average Seek Length: 32.2

Input set 7:

Track#	456	37	287	458	137	136	234	325	365	290	167	100	258	127	411	356	278
Deadline	9	1	1	8	2	8	1	8	9	8	9	10	7	5	8	9	12

Table 7.7.1: Analysis of EDF, SCAN-EDF and CSCAN for input set 7.

--

Algorithm used: EDF

Tracks to be accessed: 17

Total # Tracks Traversed: 2384

Tracks were Traversed in the following Order: 50 37 287 234 137 127 258 458 136 325 287 411 456 365 167 356 100 278

Tracks that are not completed tasks: 287 234 411 365 167 356

Average Seek Length: 139.88235

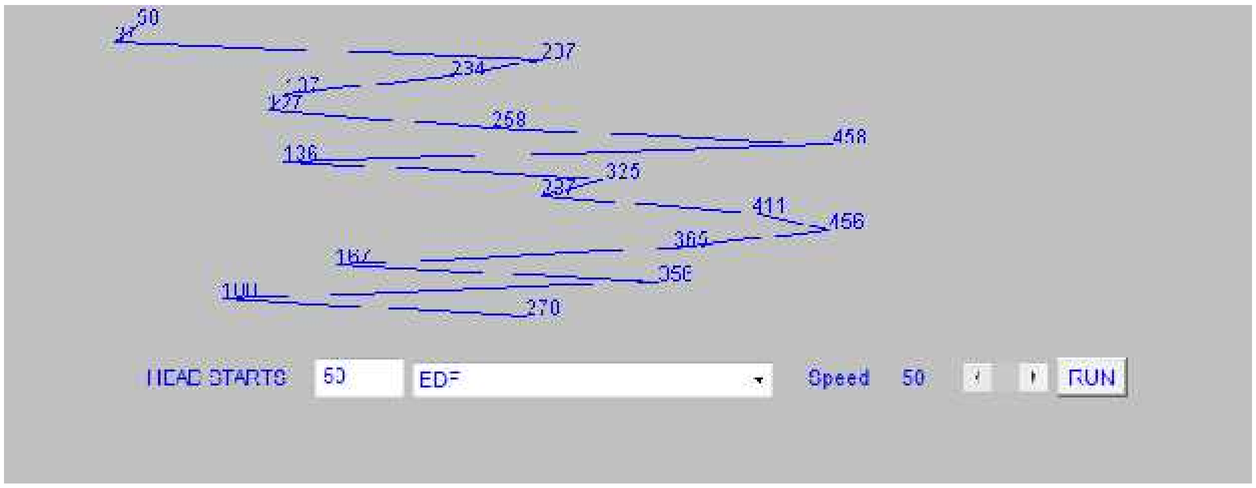


Figure 7.7.1: Execution of EDF for input set 7

Algorithm used: SEDF

Tracks to be accessed: 17

Total # Tracks Traversed: 2112

Tracks were Traversed in the following Order: 50 37 234 287 137 127 258 136 287 325 411 458 167 356 365 456 100 278

Tracks that are not completed tasks: 234 287 458 356 365 456

Average Seek Length: 124.23529

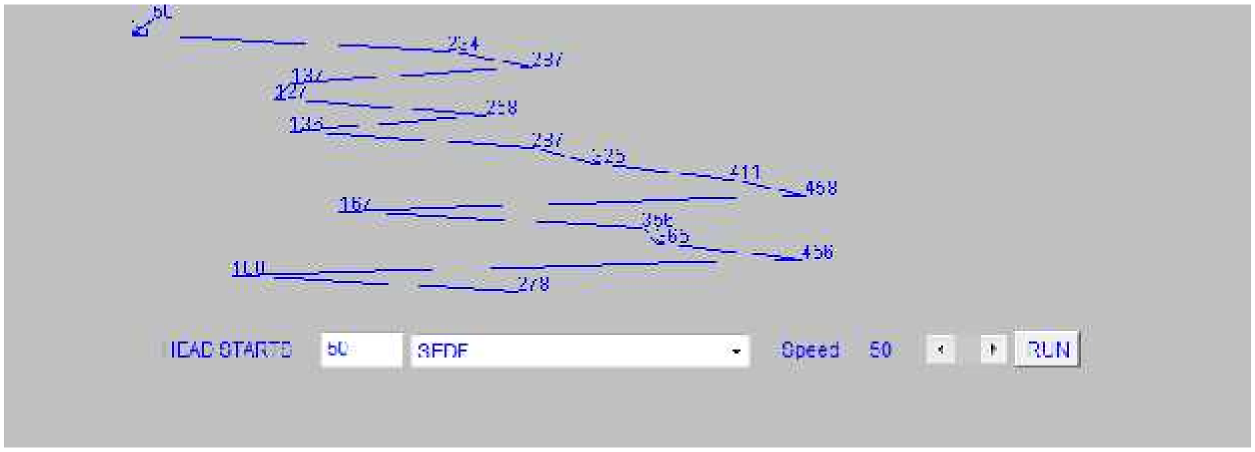


Figure 7.7.2: Execution of SCAN-EDF for input set 7

Algorithm used: CSCAN

Tracks to be accessed: 17

Total # Tracks Traversed: 736

Tracks were Traversed in the following Order: 50 100 127 136 137 167 234 258 278 287 325 356 365 411 456 458 499 0 37 287

Average Seek Length: 28.588236

Input Set 8:

Track#	256	134	256	178	296	375	421	26	345	267	100	198	489	415	321	236	149	16	59	200
Deadline	5	6	5	7	9	7	8	6	7	8	7	8	9	6	8	7	9	6	8	6

Table 7.8.1: Analysis of EDF,SCAN-EDF and CSCAN for input set 8.

Algorithm used: EDF

Tracks to be accessed: 20

Total # Tracks Traversed: 3601

Tracks were Traversed in the following Order: 50 256 256 134 26 415 16 200 178 375 345 100 236 421 267 198 321 59 296 489 149

Tracks that are not completed tasks: 200 375 345 100 236 267 198 321 59 489 149

Average Seek Length: 180.05

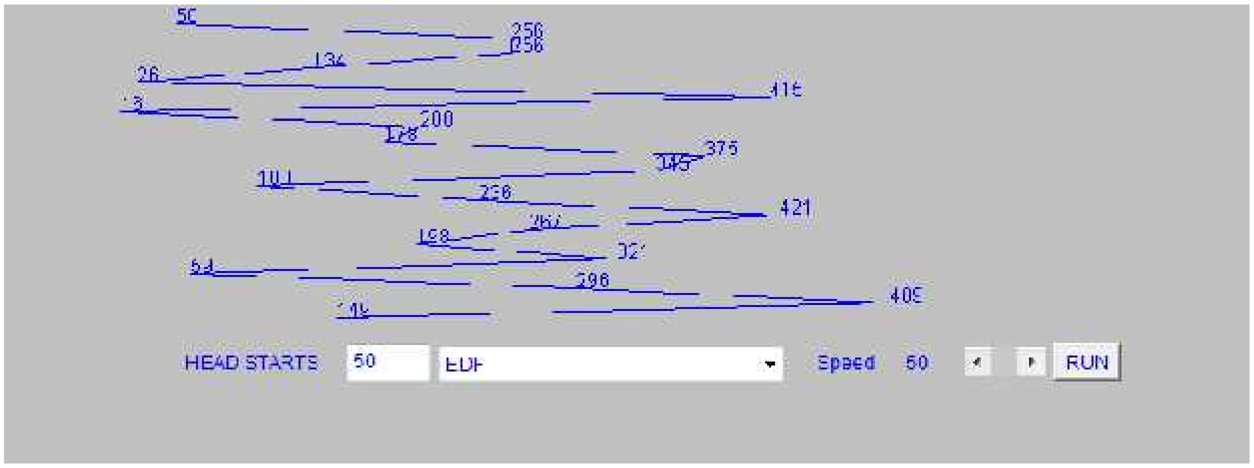


Figure 7.8.1: Execution of EDF for input set 8

Algorithm used: SEDF

Tracks to be accessed: 20

Total # Tracks Traversed: 2725

Tracks were Traversed in the following Order: 50 256 256 16 26 134 200 415 100 178 236 345 375 59 198 267 321 421 149 296 489

Tracks that are not completed tasks: 415 178 236 345 375 198 267 321 421 296 489

Average Seek Length: 136.25

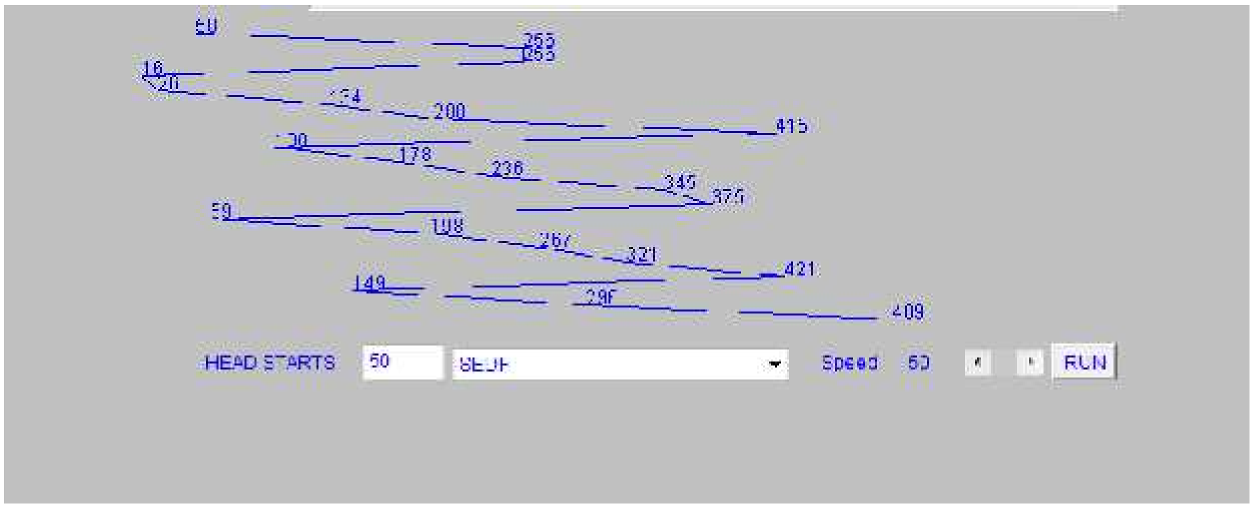


Figure 7.8.2: Execution of SCAN-EDF for input set 8

Algorithm used: CSCAN

Tracks to be accessed: 20

Total # Tracks Traversed: 705

Tracks were Traversed in the following Order: 50 59 100 134 149 178 198 200 236 256 267 296 321 345 375 415 421 489 499 0 16 26 256

Average Seek Length: 35.25

Input set 9:

Track#	60	80	75	110	210	150	65	175	200	190	185	250	350	220	195	380	420	400	370	450	360	34
Deadline	2	5	6	3	3	8	3	6	1	8	6	3	2	7	1	4	5	7	8	7	2	6

Table 7.9.1: Analysis of EDF, SCAN-EDF and CSCAN for input set 9.

--

Algorithm used: EDF

--

Tracks to be accessed: 22

Total # Tracks Traversed: 2512

Tracks were Traversed in the following Order: 50 80 65 250 420 360 60 195 34 75 210 200 220 400 150 175 190 370 110 185 350 380 450

Tracks that are not completed tasks: 195 350 360 210 65 250 420 175 185 34 400 450 190 370

Average Seek Length: 162.81818

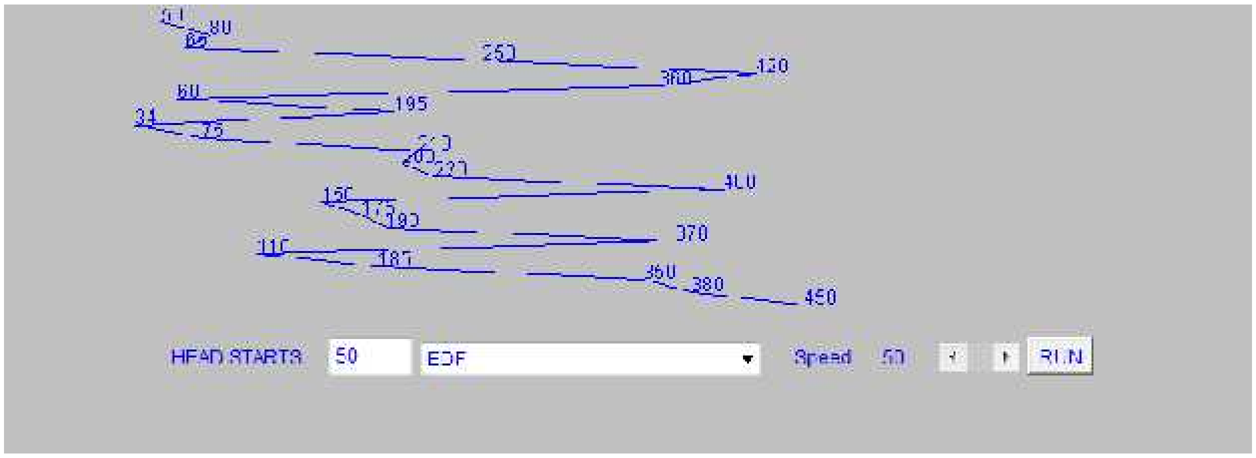


Figure 7.9.1: Execution of EDF for input set 9

Algorithm used: SEDF

Tracks to be accessed: 22

Total # Tracks Traversed: 2432

Tracks were Traversed in the following Order: 50 65 80 250 360 420 34 60 195 75 200 210 220 400 150 175 190 370 110 185 350 380 450

Tracks that are not completed tasks: 200 350 360 110 210 250 420 75 175 185 400 450 190 370

Average Seek Length: 143.72728

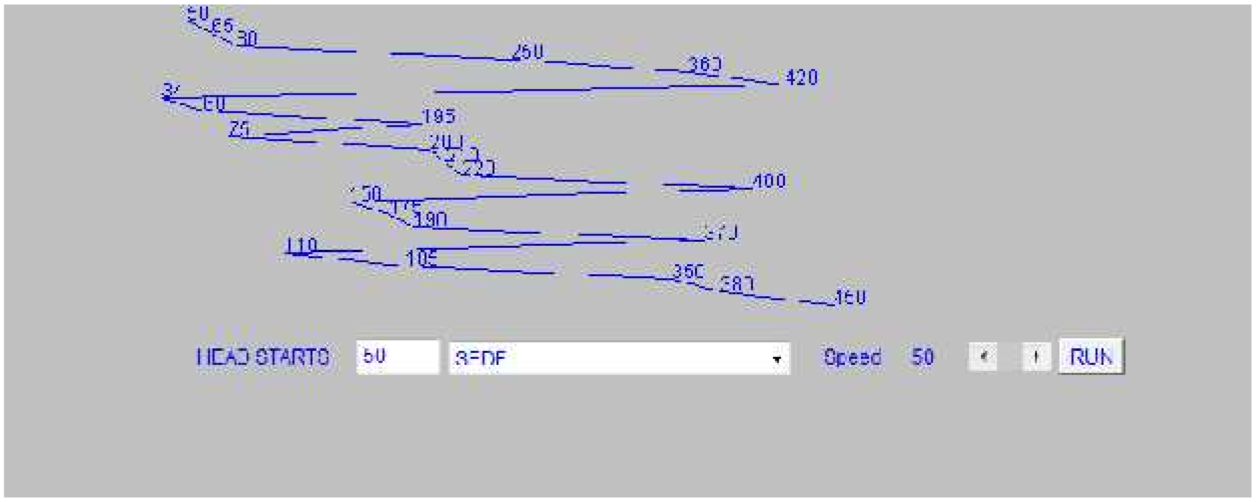


Figure 7.9.2: Execution of SCAN-EDF for input set 9.

Algorithm used: CSCAN

Tracks to be accessed: 22

Total # Tracks Traversed: 483

Tracks were Traversed in the following Order: 50 60 65 75 80 110 150 175 185 190 195 200 210 220 250 350 360 370 380 400 420 450 499 0 34

Average Seek Length: 21.954546

No. of Input		3	5	8	10	12	15	17	20	22
Seek Length	**SCAN-EDF**	145	83.6	131.50	165.1	135	130.26	124.23	136.25	143.72
	EDF	145	99.6	132.0	168.7	165.33	147.20	139.88	180.05	162.81
	CSCAN	149.66	99.0	62.0	45.30	38.16	32.2	28.588	35.25	21.95

Table 7.1.1 (a): Seek Length of SCAN-EDF, EDF and CSCAN

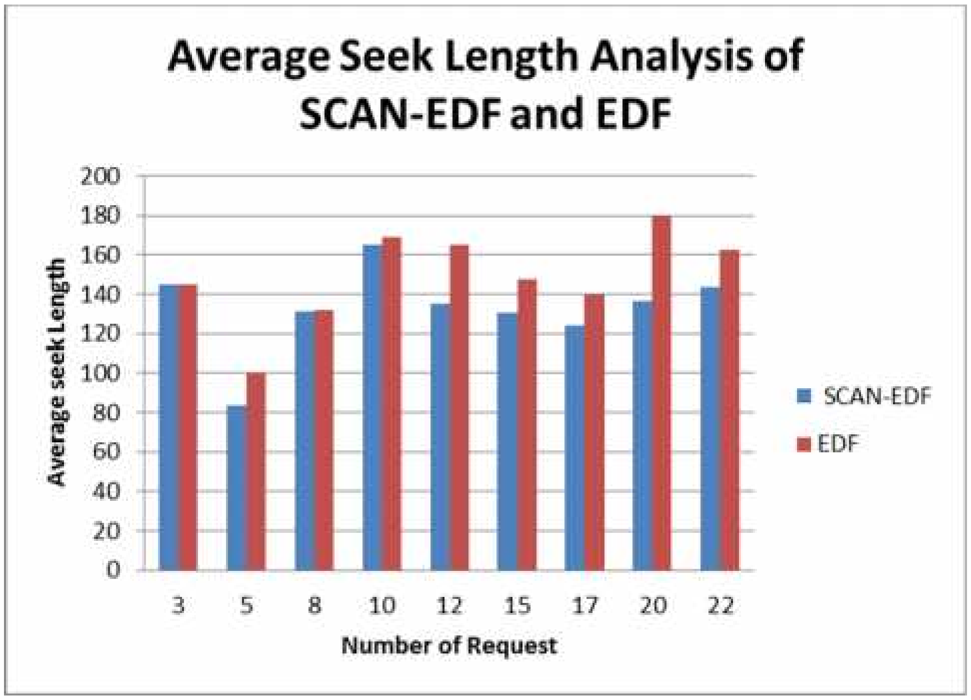


Figure 7.1.1 (a): Seek Length Analysis of SCAN-EDF and EDF on different number of Requests

No. of Input		3	5	8	10	12	15	17	20	22
Track Traversed	SCAN-EDF	435	418	1052	1651	1620	1954	2112	2725	3162
	EDF	435	498	1056	1687	1984	2208	2378	3601	3582
	CSCAN	449	495	496	453	458	483	486	705	483

Table 7.1.1(b): Track Traversed by SCAN-EDF, EDF and CSCAN under varying request

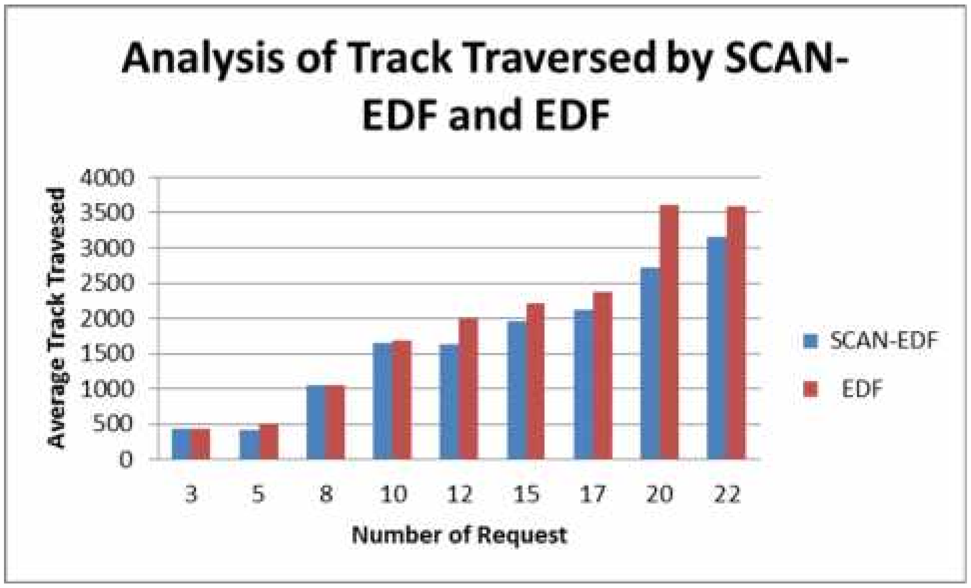


Figure 7.1.1(b) Track Travesrsed Analysis of SCAN-EDF and EDF on different number of Requests.

7.2 Experimental Results

Fig.7.1.1 (a) shows that the total seek length of the disk scheduling algorithm SCAN-EDF and EDF. We have found that the seek length of the SCAN-EDF is less than the EDF as the number of requests increases which effects the performance of the system.

Also Fig. 7.1.1 (a) shows that the SCAN-EDF has performance very close to the EDF under the low number of requests.

Seek length of the CSCAN is less than the SCAN-EDF and EDF but this algorithm do not account for the time or deadlines in scheduling decisions. So CSCAN algorithm is not suitable for Real-Time System. It is seen that the seek length of the CSCAN is more in the case of less number of requests and seek length of CSCAN decreases while the work load increases.

Fig. 7.1.1(b) shows that the number of track traversed by the SCAN-EDF, EDF and CSCAN. From this figure, we can see that the number of track traversed by the SCAN-EDF is less than the EDF.

During testing of the different algorithms, it is seen that the SCAN-EDF performance increases in the case of number of various requests having the same deadlines than the EDF. It means that the efficiency of the SCAN-EDF also depends upon the number of requests having the same deadline. If all the tasks have not near deadlines or all the tasks have different deadlines, the scheduling result of SCAN-EDF is same as EDF.

So from the above all observations, we derived that the proposed algorithm SCAN-EDF have better performance than the EDF and CSCAN because seek length is less, miss rate is less and number of track traversed is less than the other two algorithms EDF and CSCAN.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this dissertation, the comparison behavior of the two most famous disk scheduling algorithms that schedule the order of disk I/Os which are used today for developing real time disk scheduling algorithms SCAN-EDF, EDF and CSCAN have been done. The comparison analysis is done with the help of simulator which is constructed in the Java Programming Language.

We found that the disk scheduling algorithm SCAN-EDF was found to produce less seek length, less number of track traversed by the number of requests than the EDF due to lower arm utilization.

8.2 Further Recommendation

Most of the scheduling techniques described in this dissertation are based on a single disk so the extension to our work can be that such algorithm can be done for disk array (RAID).The performance differences between SCAN-EDF and EDF are higher with an array than with a single disk. This improvement is primarily due to the fact that seek time is bigger fraction of the service time in the array.

References

[1] A. L. N Reddy Texas A & M University Jim Wyllie Disk Scheduling in a multimedia I/O System IBM Almaden Research Center and K.B.R WIJAYARATNE Sanp Appliance Inc. (Division of Adaptec Inc.) ACM Transactions on Multimedia Computing, Communication and Application February 2005.

[2] A.L Narasimha Reddy and james C. Wyllie I/O Issues in a Multimedia System, IBM Almaden Research Center 1994 IEEE Journal

[3] S. Baskiyar, N. Meghanathan, A Survey of Contemporary Real-Time Operating Systems, Informatica 29 (2005) 233-240.

[4] Buddhikot MM, Parulkar GM, Cox JR (1994) Design of a large scale multimedia storage server. J Comput Netw ISDN Syst 27(3):503–517

[5] Kopetz, H., Real-Time Systems, Design Principles for Distributed Embedded Applications, Klower Academic Publishers, 1997, Chpt. 10-11.

[6] Plagemann T, Goebel V, Halvorsen P, Anshus O (2000) Operating system support for multimedia systems. Comput Commun J 23(3):267–289

[7] Shenze Chen, James F. Kurose, John A. Stankovic, Don Towsley Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems

[8] Sameh M. I. E, a real-time disk scheduling algorithm for multimedia storage servers Alexandria University, 1996

[9] H. Rahmani, M. R. Bonyadi, A. Momeni. Mohsen Ebrahimi Moghaddam. Maghsoud Abbaspour Hardware design of a new genetic based disk scheduling method Springer real time systems 2011.

[10] Gemmell J, Christoduoulakis S (1992) Principles of delay sensitive multimedia data storage and retrieval.ACM Trans Inf Syst 10(1):51–90

[11] Thomas S, Seshadri S, Haritsa JR (1996) Integrating standard transactions in firm real-time database systems.Inf Syst 21(1):3–28

[12] Hofri M (1980) Disk scheduling: FCFS vs. SSTF revisited. Commun ACM 23(11):645–653

[13] Yu PS, Chen MS, Kandlur DD (1992) Design and analysis of a grouped sweeping scheme for multimedia storage management. In: 3rd international workshop on network and operating system support for digital audio and video, SanDiego, California, pp 44–55

[14] Yu PS, Chen MS, Kandlur DD (1993) Grouped sweeping scheduling for DASD-based multimedia storage management. Multimed Syst 1(3):99–109

[15] Gemmell J, Christoduoulakis S (1992) Principles of delay sensitive multimedia data storage and retrieval.ACM Trans Inf Syst 10(1):51–90

[16] Sohn JM, Kim GY (1997) Earliest-deadline-first scheduling on nonpreemptive real-time threads for continuous media server. In: Proceedings of the conference on high-performance computing and networking

[17] Abbott R, Molina HG (1990) Scheduling I/O requests with deadlines: a performance evaluation. In: Proceedings of the IEEE real-time systems symposium, RTSS, pp 113–124

[18] Tanenbaum AS (2001) Modern operating systems, 2nd edn. Prentice Hall, New York

[19] Clifford W. Mercer Nov 13 1992 An, Introduction to Real-Time Operating Systems: Scheduling Theory

[20] Authors: Kanaka Juvva Real-Time Systems Carnegie Mellon University 18-849b Dependable Embedded Systems Spring 1998

[21] Klara. N, & Jonathan, M. Smith (1995).The QoS Broker.IEEE Multimedia Magazine. Spring 1995 2(1),(pp. 53-67).

[22] Deitel, An Introduction to Operating Systems, Second Edition, Addison-Wesley Publishing Company.

[23] H. Vin, A. Goyal, and P. Goyal, Algorithms for Designing Large-Scale Multimedia Servers, Computer Communications, Vol. 18, No. 3, 1995

[24] IEEE Information technology - Portable Operating System Interface (POSIX) - Part 1: Base Definitions; Part 2: System Interfaces; Part 3: Shell and Utilities; Part 4: Rationale.

http://standards.ieee.org/catalog/olis/posix.html

[25] http://www.fsmlabs.com

[26] IEEE Information Technology – Portable Operating System Interface (POSIX): IEEE/ANSI Std 1003.1, 1996 Edition.

[27] K. Lin, Y. C. Wang, "The Design and Implementation of Real-Time Schedulers in RED-Linux", Proceedings of the IEEE, Vol. 91, No. 7, July 2003.

[28] W. Dinkel, D. Niehaus, M. Frisbie, J. Woltersdorf, KURT-Linux User Manual, Information and Telecommunication Technology Center, University of Kansas, 2002.

[29] Cheng-Han Tsai, Edward T. H Chu and Tai Yi Huang A Rate-Based Real-Time Disk Scheduling Algorithm ACM 2004.

[30] Xu. J. & Parnas, D. (1990). Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations. IEEE Trans. On Software Engineering. Vol.16 (3). (pp. 360-369).

[31] S.Y Amdani, M.S Ali, S.M Mundada Mathematical Model for Real Time Disk Scheduling Problem Emerging Trends in Computer Science and Information Technology-2012.

Appendix

DiskApplet.java

```
import java.awt.*;
import java.io.*;
import java.util.*;
import java.applet.Applet;
/*
        <applet code="DiskApplet" width=300 height=200>
        </applet>
*/

public class DiskApplet extends Applet
{

        public TextArea sumTA, outTA , inputTA; // TO DISPLAY THE OUTPUT
        public static TextArea outTF = null;
        public TextArea pageTA = null; // TO GET THE INPUT - PAGES REQ
        public static TextField jobTF = null;
        public TextField headTF, sectTF , suTF; // INPUT
        public static  Choice alg;
        public TrackCanvas outC;
        public int rand = 0;
        public String page_request = null;
        public  Vector heightV = null, sectV = null ;
        public static int sectorUsed =0;
        public int sectors = 0, start =100, jobs = 0;
        private static DScheduler sch = null;
        private TrackCanvas r ;
        Thread Animation_thread;

        Scrollbar barSpeed;
        Label labelSpeed;
        Label labelSpeedStatus;
        static int nSpeed = 50;

        public int calcBarHeight(){
                sectors = 200;
                int height = 0;
                height = sch.sectorUsed/10 ;
                return height;
```

```
	}
	public int calcBarSect(){
		return sectorUsed;
	}
	public static void dispOutput( ){

		sectorUsed = sch.sectorUsed ;
		//outC.drawOutput( start, jobs, sectors, sch.getResult());
		outTF.append("\n-------------------------------------------------------------------------
");
		outTF.append("\nAlgorithm used:   " + alg.getSelectedItem() +"\n");
		outTF.append("--------------------------------------------------------------------------");
		outTF.append("\nTracks to be accessed: " + getIntFromTF(jobTF) + "\tTotal #
Tracks Traversed: " + sectorUsed + "\n");
		outTF.append("Tracks were Traversed in the following Order: " +
sch.getResultString() + "\n");
		outTF.append("Tracks that are not completed tasks:  " + sch.rmvS +"\n");
		outTF.append("Percent of task completion within deadlines:  " +
sch.preTrackFinished +"\n");
		outTF.append("Average Seek Length:   " +
(float)sectorUsed/(float)getIntFromTF(jobTF) + "\n");
		outTF.append("------------------------------------------------------------------------");

	}
	public static int getIntFromTF(TextField t ){
		String text = t.getText();
		if( text.length() > 0 ){
			try{
				int i= new Integer(text).intValue();
				if( i <= 0 ) return 0;
				else return (i/2);
			}catch (NumberFormatException n){
				return 0;
			}
		}else return 0;
	}
	public int getSchType( ){

		if( alg.getSelectedItem().equals("SSTF") )
			return 0;
		else if( alg.getSelectedItem().equals("CSCAN") )
			return 1;
		else if( alg.getSelectedItem().equals("SCAN") )
```

```
                return 2;
        else if( alg.getSelectedItem().equals("CSEDF") )
                return 3;
        else if( alg.getSelectedItem().equals("SEDF") )
                return 4;
        else if(alg.getSelectedItem().equals("EDF"))
                return 5;
        else return -1;
}
public boolean handleEvent( Event event ){
        if( event.id == Event.WINDOW_DESTROY ){
        //      dispose( );
                System.exit(0);
                return true;
        }else if( event.id == Event.ACTION_EVENT){
                if( event.target instanceof Button ){
                        if( processButtons((String)event.arg) )
                                return true;
                }

        }
        if(event.target == barSpeed)  {
                        nSpeed = barSpeed.getValue()+1;
                        labelSpeedStatus.setText(String.valueOf(nSpeed-1));
                        return true;

            }
        return false ;
}
public void init( ){
        setBackground(java.awt.Color.lightGray);
        setSize(800,550);
        setForeground(java.awt.Color.blue);
        setLayout(new BorderLayout());
        setup();
        loadHeightV();
}
public void loadHeightV( ){
        heightV = new Vector( );
        sectV = new Vector( );
        for(int i=0; i<10; ++i){
                heightV.addElement( new Integer(0) );
                sectV.addElement( new Integer(0) );
```

```
			}
		}
		public boolean processButtons( String s ){
			if("Run".equalsIgnoreCase(s)){
				Animation_thread = new Thread(r, "Animation_thread");
				Animation_thread.start();
				  if( pageTA != null )
				    try{
					jobs = getIntFromTF(jobTF);
					if( jobs <= 55 ){
						start = getIntFromTF(headTF)*2;
						sectors = 500 ;
						sch = new DScheduler( getSchType(), start, sectors,
pageTA.getText() );
						page_request = pageTA.getText( );
						StringTokenizer st = new StringTokenizer( page_request );
						jobTF.setText( new Integer(st.countTokens()).toString() );
						outC.drawOutput( start, jobs, sectors, sch.getResult());
					}else pageTA.setText(" TOO HOT TO HANDLE \n Reduce the no
of Jobs ");
				    }catch(Exception ex){
				  }
				  else pageTA.setText(" Enter the Page request first");
				return true ;
			}
			return false;
		}
		public Panel setInputPanel(Label l, TextField tf){
			Panel p = new Panel( );
	   p.setLayout(new FlowLayout(2));
			p.add( l );
			p.add( tf );
			return p;

		}
		public void setup(  ){
		      GridBagLayout gbl = new GridBagLayout( );
		      setLayout( gbl );
		      GridBagConstraints[] gbc = new GridBagConstraints[6];
		      int gridx[] = { 0, 0, 0, 0, 0, 0 };
		      int gridy[] = { 0, 1, 2, 3, 4, 5};
		      int gridwidth[] = { 1, 1, 1, 1, 1, 1};
		      int gridheight[] = { 1, 1, 1, 1, 1, 1};
```

```
                Panel p[] = new Panel[6];
                p[0]=setupTitlePanel( );
                p[1]=setupManualInputPanel( );
                p[2]=setupDispPanel( );
                p[3]=setupOutputPanel( );
                p[4]=setupCBPanel( );
                //p[5]=setupRunPanel( );

              for( int i=0; i<5; ++i ){
                            gbc[i] = new GridBagConstraints( );
                          gbc[i].fill = GridBagConstraints.BOTH;
                  gbc[i].gridx = gridx[i];
                            gbc[i].gridy = gridy[i];
                            gbc[i].gridwidth = gridwidth[i];
                            gbc[i].gridheight = gridheight[i];
                            gbl.setConstraints( p[i], gbc[i] );
                            add( p[i] );
                }

        }
        public Panel setupCBPanel( ){


                Panel headP = setupInputPanel();
                Panel pr = setupRunPanel();

                Panel p = new Panel();
                p.setLayout( new GridLayout(1,3) );

                alg = new Choice();
        alg.addItem("CSCAN");
        alg.addItem("SSTF");
        alg.addItem("SCAN");
        alg.addItem("CSEDF");
        alg.addItem("SEDF");
//      alg.addItem("C-LOOK");
        alg.addItem("EDF");

                p.add( headP);
                p.add( alg );
                p.add(pr);
                return p;
```

```
}
public Panel setupDispPanel( ){

		outTF = new TextArea(9,70);
		outTF.setEditable(false);
		Panel p = new Panel();
		p.setLayout( new FlowLayout(2));
		p.add(new Label("Report:       "));
		p.add( outTF );
		return p;
}
public Panel setupGraphPanel(Label l, Canvas gc ){
		Panel p = new Panel( );
   p.setLayout(new GridLayout(1,1));
		p.add( gc );
		return p;
}
public Panel setupInputPanel( ){
		Label headL = new Label("HEAD STARTS");


		jobTF = new TextField(" ",4);
		headTF = new TextField("50",4);

		Panel headP = setInputPanel( headL, headTF);


		Panel p = new Panel( );
		p.setLayout( new GridLayout(1,1) );

		p.add( headP );

		return p;

}
public Panel setupManualInputPanel( ){
		Panel p = new Panel();
		p.setLayout( new FlowLayout(2));
		p.add(new Label("Enter Track #'s:"));
		pageTA = new TextArea(3,70);
		pageTA.setEditable(true);
		pageTA.setText("200 171 47 126 4 124 500 76");
		p.add( pageTA );
```

```
        return p;
}
public Panel setupOutputPanel( ){

        outC = new TrackCanvas(300,200, (Applet)this);
        r = new TrackCanvas(300,200, (Applet)this);

        Label outL = new Label(" OUTPUT ");

        Panel outP = setupGraphPanel(outL, outC);
        //Panel graphP = setupGraphPanel(graphL, graphC );

        Panel p = new Panel( );

        p.setLayout(new BorderLayout());

        p.add( outP );


        return p;

}
public Panel setupRunPanel( ){

        Button runB = new Button("RUN");
        barSpeed = new Scrollbar(Scrollbar.HORIZONTAL, 50, 0, 0, 101);
        barSpeed.setUnitIncrement(10);

        labelSpeedStatus = new Label("");
        labelSpeedStatus.setText("50");
        labelSpeed = new Label("Speed");
        //Button randB = new Button("RUN RANDOM");
        Panel pa = new Panel();
        pa.setLayout( new FlowLayout(2) );
        pa.add(labelSpeed);
        pa.add(labelSpeedStatus);

        Panel pb = new Panel();
        pb.setLayout( new FlowLayout(2) );
        pb.add(barSpeed);
        pb.add( runB );

        Panel p = new Panel();
```

```
		p.setLayout( new GridLayout(1,2) );
		p.add(pa);
		p.add(pb);


		//p.add( randB );

		return p;
	}
	public Panel setupTitlePanel( ){
		Label titleL = new Label( );
		titleL.setFont( new Font( "TimesRoman", 3 , 24 ) );
		titleL.setText(" Disk Scheduling ");
		titleL.setAlignment( Label.CENTER );
		Panel p = new Panel();
		p.setLayout( new FlowLayout(1) );
		p.add( titleL );
		return p;
	}
}
```

DScheduler.java

```
import java.awt.*;
import java.util.*;

public class DScheduler {

			private Vector reqV = null;
			private Stack reqS = null;
			public Vector resultV = null;
			public Vector preV = null;
			public Vector deadV = null;
			public Vector rsltV=null;
			public String rmvS=null;
			private int limit = 0;
			private int start = 0, sectors = 0;
			public  int sectorUsed = 0;
			public  float preTrackFinished;
			private Vector delV=null;
			int delE[]=new int[30];
```

```
public DScheduler(int type, int head_start, int tot_sectors, String pageRequest){
        start = head_start;
        sectors = tot_sectors;
        loadReqV(pageRequest);
        sectorUsed = 0;
        switch(type){
                case 0:
                        doSstf();
                        break;
                case 1:
                        doCScan();
                        break;
                case 2:
                        doScan();
                        break;
                case 3:
                        doCScan();
                        break;
                case 4:
                        doSEDF();
                        break;
                case 5: doEDF();
                        break;
                default:
                        break;
        }
} // END OF CONSTRUCTOR

public void checkForObject( int i ){
                if( reqV.contains(Integer.toString(i)) ){
                        int indx = reqV.indexOf( Integer.toString(i) );
                        Object o = reqV.elementAt(indx);
                        resultV.addElement(o);
                        reqV.removeElementAt(indx);
                }
        return ;
}

public void doCLook(){
        int pos = start;
        for( int i=pos; i<=sectors; i++ ){
                checkForObject(i);
        }
        pos = 0;
        for( int i=pos; i<=sectors; i++ ){
                checkForObject(i);
        }
        findSectorsUsedScan(resultV);
}
public void doCScan(){
        int pos = start;
        for( int i=pos; i<=sectors; i++ ){
                checkForObject(i);
        }
        resultV.addElement(Integer.toString(sectors-1));
        pos = 0;
        resultV.addElement(Integer.toString(pos));
        for( int i=pos; i<=sectors; i++ ){
                checkForObject(i);
        }
        findSectorsUsedScan(resultV);
//      findSectorsUsed(resultV);
} // END OF CSCAN

public void doFcfs( ){
        for(int i=0; i<reqV.size(); ++i)
                resultV.addElement(reqV.elementAt(i));
        findSectorsUsed( resultV );
} // END OF OPTIMAL ALGORITHM

public void doLook(){
        int pos = 0;
        for( int i=start; i>=0; i-- ){
                checkForObject(i);
        }
        for( int i=pos; i<=sectors; i++ ){
                checkForObject(i);
        }
        findSectorsUsed(resultV);
}
public void doScan( ){
        int pos = 0;
        for( int i=start; i>=0; i-- ){
                checkForObject(i);
        }
```

```
		resultV.addElement("0");
		for( int i=pos; i<=sectors; i++ ){
			checkForObject(i);
		}
		findSectorsUsed(resultV);
	}

	public void doSstf(){
		int curr = start, diff = 0 ;
		while ( !reqV.isEmpty() ){
			int j = -1;
			diff = new Integer((String)reqV.firstElement()).intValue() ; j=0;
			for( int i=0; i<reqV.size(); ++i ){
				int curr_diff = new
Integer(((String)reqV.elementAt(i))).intValue() - curr;
				if( curr_diff < 0 ) curr_diff *= -1;
				if( curr_diff < diff ) {
					diff = curr_diff ; j = i;
				}
			}
		  if( j >= 0 ){
			curr = new Integer(((String)reqV.elementAt(j))).intValue() ;
			resultV.addElement( reqV.elementAt(j) );
			reqV.removeElementAt(j);  j = -1;
		  }
		}
		findSectorsUsed( resultV );
	}


	public void doSEDF(){
		int previous=start;
		int deads[]=new int[30];
		double pdeads[]=new double[30];
		double pdeads1[]=new double[30];
		int min_index=0;
		for(int i=0; i<deadV.size(); ++i){
			deads[i]=new Integer(((String)deadV.elementAt(i))).intValue();
		}

		int x=1;
		for(int i=0; i<reqV.size(); ++i){
			x=new Integer(((String)reqV.elementAt(i))).intValue();
			pdeads[i]=((double)x/1000)+(double)deads[i];
		}

		for(int i=0; i<reqV.size(); ++i){
			pdeads1[i]=pdeads[i];
		}

		for(int i=0;i<deadV.size();i++){
			double min= pdeads[i];
			min_index=i;
			for(int j=i;j<deadV.size();j++){
				if(min > pdeads[j]){
					min_index=j;
					min=pdeads[j];
				}
			}
			if(min_index!=i){
				double temp=pdeads[i];
				pdeads[i]=pdeads[min_index];
				pdeads[min_index]=temp;
			}
		}
		for(int i=0;i<reqV.size();i++){
			double search=pdeads[i];
			for(int j=0;j<reqV.size();j++){
				if(pdeads1[j]==search){
					resultV.addElement(reqV.elementAt(j));
					break;
					}
			}
		}

		for(int i=0; i< resultV.size(); ++i){
			int sectMoved = new
Integer(((String)resultV.elementAt(i))).intValue() - previous;
			previous = new Integer(((String)resultV.elementAt(i))).intValue() ;
			if (sectMoved < 0 ) sectMoved *= -1;
			sectorUsed += sectMoved;
		}

		rsltV=new Vector();
		Object copyRsV[]=new Object[25];
		resultV.copyInto(copyRsV);
```

```
        for(int i=0; i< resultV.size(); i++){
                rsltV.addElement(copyRsV[i]);
        }

        int ind;
        int y;
        int d;
        for(int i=0; i<reqV.size(); i++){
                ind=rsltV.indexOf(reqV.elementAt(i));
                d=new Integer(((String)deadV.elementAt(i))).intValue();
                if(d<ind+1){
                        rsltV.removeElementAt(ind);
                }
        }



        preTrackFinished=((float)rsltV.size()/(float)reqV.size())*100;
        rmvS=new String();

        for(int i=0;i<resultV.size();i++){
                if(!rsltV.contains(resultV.elementAt(i))){
                        rmvS= rmvS + "  "+(String)resultV.elementAt(i);
                }
        }


}
public void doCSEDF(){
        int previous=start;
        int deads[]=new int[30];
        double pdeads[]=new double[30];
        double pdeads1[]=new double[30];
        int min_index=0;
        for(int i=0; i<deadV.size(); ++i){
                deads[i]=new Integer(((String)deadV.elementAt(i))).intValue();
        }

        int x=1,y;
        for(int i=0; i<reqV.size(); ++i){
                x=new Integer(((String)reqV.elementAt(i))).intValue();
                pdeads[i]=((double)x/1000)+(double)deads[i];//perturbation of
deadlines
```

```
            }

            for(int i=0; i<reqV.size(); ++i){
                pdeads1[i]=pdeads[i];
            }

            for(int i=0;i<deadV.size();i++){
                double min= pdeads[i];
                min_index=i;
                for(int j=i;j<deadV.size();j++){
                    if(min > pdeads[j]){
                        min_index=j;
                        min=pdeads[j];
                    }
                }
                if(min_index!=i){
                    double temp=pdeads[i];
                    pdeads[i]=pdeads[min_index];
                    pdeads[min_index]=temp;
                }
            }
            for(int i=0;i<reqV.size();i++){
                double search=pdeads[i];
                for(int j=0;j<reqV.size();j++){
                    if(pdeads1[j]==search){
                        if(i==0)
                            resultV.addElement(reqV.elementAt(j));
                        x=new
Integer(((String)reqV.elementAt(j))).intValue();
                        y=new
Integer(((String)resultV.lastElement())).intValue();
                        if(x < y){
                            if(y<500){

        resultV.addElement(Integer.toString(sectors));
                            }
                            int pos = 0;
                            resultV.addElement(Integer.toString(pos));
                        }
                        if(i>0){
                            resultV.addElement(reqV.elementAt(j));
                            break;
                            }
                        }
                }
        }


        for(int i=0; i<resultV.size(); ++i){
                int curr = new Integer(((String)resultV.elementAt(i))).intValue();
                int sectMoved = 0;
                if( curr > previous )
                sectMoved = curr - previous;
                previous = new Integer(((String)resultV.elementAt(i))).intValue() ;
                if (sectMoved<0 ) sectMoved *= -1;
                sectorUsed += sectMoved;
        }

}

public void doEDF(){
        int previous=start;

        int deads[]=new int[30];
        int deads1[]=new int [30];
        int min_index=0;
        for(int i=0; i<deadV.size(); ++i){
                deads[i]=new Integer(((String)deadV.elementAt(i))).intValue();
        }
        for(int i=0; i<reqV.size(); ++i){
                deads1[i]=deads[i];
        }
        for(int i=0;i<deadV.size();i++){
                int min= deads[i];
                min_index=i;
                for(int j=i;j<deadV.size();j++){
                        if(min > deads[j]){
                                min_index=j;
                                min=deads[j];
                        }
                }
                if(min_index!=i){
                        int temp=deads[i];
                        deads[i]=deads[min_index];
                        deads[min_index]=temp;
                }
```

```
		}


		for(int i=0;i<reqV.size();i++){
			int search=deads[i];
			if(i==0){
				for(int j=0;j<reqV.size();j++){
					if(deads1[j]==search){
						resultV.addElement(reqV.elementAt(j));
					}
				}
			}
			else{ if(deads[i]!=deads[i-1]){
				for(int j=0;j<reqV.size();j++){
					if(deads1[j]==search){
						resultV.addElement(reqV.elementAt(j));
						}
					}
				}
			}
		}


		for(int i=0; i< resultV.size(); ++i){
			int sectMoved = new
Integer(((String)resultV.elementAt(i))).intValue() - previous;
			previous = new Integer(((String)resultV.elementAt(i))).intValue() ;
			if (sectMoved < 0 ) sectMoved *= -1;
			sectorUsed += sectMoved;
		}


		rsltV=new Vector();
		Enumeration resultE=resultV.elements();
		while(resultE.hasMoreElements()){
			rsltV.addElement(resultE.nextElement());
		}

		int ind;
		int y;
		int d;
		for(int i=0; i<reqV.size(); i++){
			ind=rsltV.indexOf(reqV.elementAt(i));
```

```
                d=new Integer(((String)deadV.elementAt(i))).intValue();
                if(d<ind+1){
                        rsltV.removeElementAt(ind);
                }
        }
        preTrackFinished=((float)rsltV.size()/(float)reqV.size())*100;

        rmvS=new String();

        for(int i=0;i<resultV.size();i++){
                if(!rsltV.contains(resultV.elementAt(i))){
                        rmvS= rmvS + "  "+(String)resultV.elementAt(i);
                }
        }


}


public void findSectorsUsed( Vector v ){
        int previous = start ;
        for(int i=0; i< v.size(); ++i){
                int sectMoved = new Integer(((String)v.elementAt(i))).intValue() -
previous;
                previous = new Integer(((String) v.elementAt(i))).intValue() ;
                if (sectMoved < 0 ) sectMoved *= -1;
                sectorUsed += sectMoved;
        }
}

public void findSectorsUsedScan( Vector v ){
        int previous = start ;
        for(int i=0; i<v.size(); ++i){
                int curr = new Integer(((String)v.elementAt(i))).intValue();
                int sectMoved = 0;
                if( curr > previous )
                sectMoved = curr - previous;
                previous = new Integer(((String)v.elementAt(i))).intValue() ;
                if (sectMoved<0 ) sectMoved *= -1;
                sectorUsed += sectMoved;
        }
}
```

```
        public Vector getResult( ){
                return resultV;
        }

        public String getResultString( ){
                StringBuffer sb = new StringBuffer(Integer.toString(start)+" ");
                for(int i=0; i<resultV.size(); ++i)
                        sb.append( (String)resultV.elementAt(i) + " " );
                return sb.toString();
        }

        public void completionJog(){
                int x;
                int y;
                int d;
                for(int i=0; i<reqV.size(); ++i){
                        x=new Integer(((String)reqV.elementAt(i))).intValue();
                        for(int j=0; j<reqV.size(); ++j){
                                y=new Integer(((String)resultV.elementAt(j))).intValue();
                                if(x==y){
                                        d=new
Integer(((String)deadV.elementAt(i))).intValue();
                                        if(i<j){
                                                if(d-j+i<0){
                                                        delE[i]=j;
                                                }

                                        }
                                        else{
                                                if(d-j-1<0){
                                                        delE[i]=j;
                                                }
                                        }
                                }
                        }
                }
        }

        public void loadReqV(String s){

                reqV = new Vector();
                deadV=new Vector();
                preV=new Vector();

                reqS = new Stack();
                resultV = new Vector();
                StringTokenizer st = new StringTokenizer(s,"\n");

                while( st.hasMoreTokens() )
                        preV.addElement( st.nextToken() );

                String s1,s2;
                s1=(String)preV.elementAt(0);
                s2=(String)preV.elementAt(1);

                StringTokenizer st1 = new StringTokenizer(s1);
                StringTokenizer st2 = new StringTokenizer(s2);

                while( st1.hasMoreTokens() )
                        deadV.addElement( st1.nextToken() );
                while( st2.hasMoreTokens() )
                        reqV.addElement( st2.nextToken() );

                for(int i=reqV.size()-1; i>=0; --i)
                        reqS.push(reqV.elementAt(i));

        } // END OF LOADING PAGE REQUEST IN A VECTOR & IN A STACK
} // END OF CLASS SCHEDULER
```

TrackCanvas.java

```
import java.awt.*;
import java.util.Vector;
import java.applet.Applet;

public class TrackCanvas extends Canvas implements Runnable{

        public  int width=0, height=0 ;
        private Vector inputV =null;
        private int start =0, x_incr =0, y_incr =0;
        private Image img = null ;
        public Applet a = null;

        public TrackCanvas(int w, int h, Applet a){
```

```
                        super();
                        width = w; height = h;
                        this.a = (Applet)a;

                        img = a.getImage(a.getCodeBase(), "dum.gif");
                        setSize(width,height);
                        loadV();
                        repaint( );

                }
                public void drawOutput( int start, int jobs,  int sectors,  Vector v ){
                        inputV = v;
                        int nojobs = v.size() + 2 ;
                        x_incr = width / sectors ;
                        y_incr = height / (nojobs) ;
                        this.start = start ;
// System.out.println(" The xincr " + x_incr + " and the yincr " + y_incr );
                        repaint( );

                }
                public void loadV(){
                        inputV = new Vector();
                }
                public void loadV(int start, int jobs,  int sectors,  Vector v ){
                        inputV = new Vector();
                }
                public void paint( Graphics g ){

                        int x1=0, x2=0, y1=0, y2=0;
                        x2 = start + x_incr; y2 = y_incr;
                //      g.drawLine( x1, y1, x2, y2 );
                        g.drawString(Integer.toString(start), x2, y2 );
                        for( int i=0; i<inputV.size(); ++i){
//                              x1 = new Integer(((String)inputV.elementAt(i))).intValue() +
x_incr;
//                              y1 += y_incr;
                                x1 = x2; y1 = y2;
                                x2 = new Integer(((String)inputV.elementAt(i))).intValue() +
x_incr;
                                y2 += y_incr;
                                g.drawImage( img, x1-10, y1, a );
                                g.drawString((String)inputV.elementAt(i), x2, y2  );
                                g.drawLine( x1, y1, x2, y2 );
```

```
                    //          System.out.println(" Line drawn at cord - " + x1 + " " + y1 + " " +
x2 + " " + y2 + " " );
                                try{Thread.sleep(10000L/DiskApplet.nSpeed);}
                    catch(InterruptedException e){//do nothing
                    }
                    }
                    g.drawImage(img, x2, y2, a );
                    DiskApplet.dispOutput( );


               }
 public void run(){


 }
}
```