Fairness in Disk Scheduling

A Project Report
Submitted in partial fulfilment of the
requirements for the Degree of
Master of Engineering
in
Computer Science and Engineering

by

Raj Kumar

Department of Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012

January 1993

Acknowledgements

I take this opportunity to thank my project supervisor, **Dr. Jayant R. Haritsa**, Assistant Professor, Department of Supercomputer Education & Research Centre, Indian Institute of Science(IISc), Bangalore, for his excellent guidance, sound advice and persistent encouragement throughout the course of this work. I am highly indebted to him for his unstinting help without which this project would have never materialized.

I am grateful to the Chairman of our department, *Prof N. Viswanadham*, for allowing me to work in SERC department for my project work. I am thankful to *Dr. Mathew Jacob & Prof. Y. Narahari* of our department and *Mr. R. Krishnamurthy* of SERC dept. for providing valuable suggestions at various stages of my project work. I would also like to thank all the staff members of CSA department for their cooperation and those who helped me in one way or the other during my stay in IISc.

During my stay in the institute, I counted on many good friends, particularly Sanjay, Jonny, Gaur, Malay, KSub, and my room-mate Hemant, whose accompany made my stay in IISc an enjoyable and unforgettable experience.

Finally, and most importantly, I thank my *parents & grand parents* for their help, understanding, and blessings without which I would have never made it to this prestigious institute in our country. Their love and constant encouragement were a perpetual source of inspiration to me.

Abstract

The techniques used by a computing system to handle its secondary memory requirements greatly influence the quality of service provided by it. In an interactive multi-user computing environment, it is not only necessary to improve throughput but it is equally important to give each user a *fair* share of the system resources. By *fairness*, we mean that if there are n users competing for m resources, then each user should get $\frac{1}{n}$th of each of these resources. Researchers have done good job of finding policies for CPU scheduling that give *fair* treatment to all users without degrading throughput considerably. However, people have looked from the point of reducing *average response time*, *average seek time*, *average waiting time* etc., as far as disk system is concerned. Not much has been done about *fairness* at disk level. We believe that *fairness* at disk level is equally important.

In this thesis, we study various disk scheduling algorithms and analyse how *fair* they are by using simulation. We quantify each algorithm by *fairness* performance metric and study how each algorithm behaves under various loading conditions. Also, we developed a disk scheduling algorithm named *Non-Stop Circular Polling* which is a variant of the classical *Circular Polling* algorithm. Our algorithm is *fair* under all the work loads we considered.

Contents

Chapter 1

INTRODUCTION

1.1 Motivation

Assume, for the moment, that you are a teller in a bank, serving a long queue of customers standing in front of your counter. Consider the situation where one of your close friends is standing somewhere back in the queue. Also assume that time is nearing lunch-break time in your office and you want to take your friend to lunch with you. Being human beings, there is always a tendency to favor persons near and dear to you. So, would you invite your friend to the front of queue so that as soon as his monetary transaction is over, you both can break for lunch or would you let your friend stand in the queue. You, the teller in a bank, have just been faced with one of the important problems in computer systems – scheduling a set of tasks so as to optimize some metric; in the present situation, whether to be *fair* to everyone in the queue or discriminate against the customers ahead of your friend by calling him to the front of the queue. In this dissertation, we address such scheduling problems in the context of a disk system. In the disk system context, customers standing in a queue play the role of requests for disk service; you, the teller, play the role of read/write head and the box containing money to complete monetary transactions plays the role of a disk.

1.2 General overview

Scheduling is a fundamental operating system concept, since almost all computer resources are scheduled before use. One of the important areas of research that has received considerable attention of the computer scientists and researchers in last few decades is *multiprogramming*, i.e. allowing more than one program to execute at the same time. These concurrently executing programs place varying degree of requirements on computer resources. Scheduling these concurrently executing programs on computer resources so as to optimize some performance metric is an important problem in Computer Science.

A typical computer system consists of a central processing unit(CPU), the memory buffers, and disk as physical resources. CPU being one of the important resources in a computer system has received considerable attention as far as scheduling is concerned. Various CPU scheduling algorithms have been proposed in literature such as First Come First Served, Shortest Job First, Shortest Remaining Job First, Round Robin scheduling, Priority based scheduling, Multi-Level Queues, Multi-Level Feedback Queues, etc.[16], [18]. These algorithms try to optimize one or the other performance metric. Some of the important performance metrics [18] that people have looked at as far as CPU scheduling is concerned are as follows:

- Efficiency: Keep the CPU as busy as possible.
- Response time: Minimize the response time for interactive users.
- Turnaround time: Minimize the waiting time of batch users for output.
- Throughput: Maximize the number of jobs processed per unit of time.
- Fairness: Each process gets its fair share of CPU.

However, all these performance metrics are not independent of each other. For example, throughput and fairness are two conflicting goals of performance to achieve. That is, if you try to maximize throughput by scheduling shorter jobs first in preference to longer ones, you become unfair to longer jobs [18]. On the contrary, if you try to be fair, the throughput will suffer. *Fairness* is an important performance metric for multi-user interactive computer systems like the systems running under UNIX operating system. Over the years, people have looked at the fairness problem as avoiding *starvation*. But, in a time shared system, each user expects response from the system in a reasonable amount of time. It will not be acceptable to any user if he/she receives unfair treatment from the system. Many fair algorithms for CPU scheduling have been proposed in literature [14], [2]. UNIX uses *Round Robin Multi-Level Feedback Queues* [1] to provide fair service to multiple users while not seriously degrading throughput.

However, CPU is not the only important resource in a computer system. Disk used as a secondary storage device is an equally important resource. Most of the processing of modern computer systems such as data intensive applications center around disk system. Main memory being costly and limited in size cannot store all programs and data required for the successful completion of an application. Disks provide the primary on-line storage of information, both programs and data. Most programs, like compilers, assemblers, editors, formatters, database management systems and so on, reside on disk until loaded into the main memory for execution. These programs use disk as both the source and destination of their processing. Hence proper management of disk storage is of central importance to a computer system [16].

Physically, a disk has flat circular shape like a phonograph record. Disk surfaces are logically divided into *tracks* and information is stored on disk by magnetically recording it on the track currently under read/write head as shown in figure 1.1. Tracks are further divided into *sectors* to make the process of storing and reading information from a disk efficient. When a number of disks are mounted on a spindle, the corresponding tracks of each disk put together are referred to as *cylinders*. Whenever data is to be accessed from a disk, the read/write head reads data from the specified tracks and transfers it.

Various scheduling algorithms such as First Come First Serve(FCFS), Shortest Seek Time First(SSTF), SCAN, N-Step SCAN, Circular SCAN [20], Polling [5], Circular Polling, System V Standard (SVS) [10] etc. have been proposed in literature for disk scheduling. People have looked at *expected seek time, mean response time, mean queue length, expected*

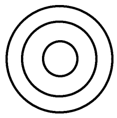

Figure 1.1: A single disk

waiting time of individual requests and *variance of waiting time* as the performance criteria for disk scheduling algorithms. In an operating system like UNIX, users have no control over the placement of their files and data on disk. However, from a user's perspective, it does not matter where his files are kept on disk by the operating system. A user's concern is that he should not be discriminated against by the system. Our objective in this study is to analyse how fair the existing disk scheduling algorithms are. What we are looking for is a plot of *average response time versus cylinder number* such as the one shown in figure 1.2 to capture the fairness property of an algorithm.

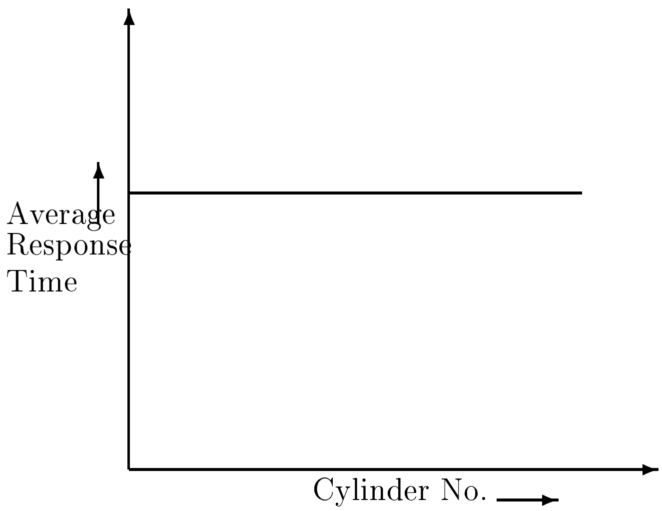

Figure 1.2: An Ideal Fair Disk Scheduling Algorithm

However, we found that average response time versus cylinder number curves for the existing disk scheduling algorithms deviate from the one shown in figure 1.2. These curves have lower average response time for the cylinders in the middle than for the cylinders on the extreme ends of the disk. Due to this behaviour of these algorithms, the users whose data is placed on the middle of the disk get better response than the ones whose data is placed on the outer cylinders.

Naturally, one will ask, is it possible to use one of the fair CPU scheduling algorithms for disk scheduling to provide fair treatment to all users? The answer is no, because disk is an inherently unfair resource since read/write head has to physically move from one cylinder to next for serving a request. The distance that head needs to travel before serving a request depends both on its present position and the cylinder having a request to be serviced. Therefore, the *context switch time* in this case is not constant unlike the CPU case where it is almost constant. Also, it is not possible to provide instantaneous fairness in a disk system because unit of data transfer to and from a disk is a **block**. One has to look at statistical fairness over a large number of requests. We have developed a new *Non-Stop Circular Polling* algorithm for disk scheduling which approximates the curve shown in figure 1.2 for various loading conditions we tried.

1.3 Organization of Dissertation

Our goal in this thesis is to profile the performance of some of the existing disk scheduling algorithms. For this, we developed an **open** and a **closed system model** of a disk and implemented a simulator based on these models. We experimented with some of the existing algorithms under a variety of loading conditions. The remainder of this dissertation is organized in the following fashion: Chapter 2 is devoted to a survey of published research on disk scheduling. Chapter 3 includes an overview of methodology, performance metrics and process of generating workload used in simulation studies. Chapter 4 covers the experimental results of open model. These results are analysed and compared with the existing ones. Also, we describe our *Non-Stop Circular Polling* algorithm in the open model context and compare its results with the ones obtained above. Chapter 5 profiles the results of all algorithms covered in Chapter 4 in the closed model context. Finally, Chapter 6 summarizes our study and outlines future research directions.

Chapter 2

OVERVIEW OF EXISTING DISK SCHEDULING ALGORITHMS

2.1 Introduction

Several disk arm scheduling algorithms have been proposed in literature for handling a queue of requests on a movable head disk such as First Come First Serve (FCFS) [20], Shortest Seek Time First (SSTF) [20], the SCAN algorithm [20], Polling [5], Circular SCAN (C-SCAN), Circular Polling [16] and so on. Each of these algorithms tries to optimize one or the other performance criteria and each has its own merits and demerits. In this chapter, we will briefly survey the existing disk scheduling algorithms.

2.2 Survey of Existing Disk Scheduling Algorithms

Existing disk scheduling algorithms that we came across in literature during the project work are listed as follows:

- **First Come First Serve (FCFS)**: FCFS serves the requests from a queue according to the order in which these requests joined the queue. FCFS does not perform any optimization. It has a random seek pattern. It does not take advantage of the positional relationships between I/O requests in the current queue or of the current cylinder position of the read/write head [20].

- **Shortest Seek Time First (SSTF)**: To serve a queue of requests using this algorithm, the queue is searched to find a request nearest to the current read/write head position. This request is the one to be served next. SSTF takes advantage of the positional relationship between the current read/write head position and I/O requests to improve the throughput, but it is at the expense of discrimination against individual requests [20]. At low arrival rate, SSTF results in random read/write head movement and essentially behaves as FCFS. At high arrival rate, SSTF tends to bias against the requests to the extreme cylinders because when arrival rate is high, it can immediately find a request near to the position where it served a request last and thus head tends to be confined to the middle of the disk most of the time. Therefore, the requests to the extreme cylinders have more waiting time and hence more response time.

- **Polling**: In Polling algorithm, the read/write head moves from the inner end of the disk to the outer end serving all requests for a cylinder as it passes by. On reaching the other end, the direction of read/write head movement is reversed [10]. Disk read/write head scheduling using Polling algorithm discriminates against requests to the extreme cylinders because two full sweeps of the disk surface occur between servicings of an extreme cylinder, while only two half sweeps occur between servicings of a central cylinder.

- **SCAN**: SCAN serves the requests from a queue as follows. Read/write head moves from inner end to the outer end of the disk servicing all the requests for a cylinder as it passes by but, when there is no pending request in the current direction, head immediately reverses its direction without going to the extreme end in the current direction. SCAN was originally proposed [7] as an alternative to SSTF to reduce the discrimination against requests to the extreme cylinders that occurs with SSTF. This algorithm reduces the variance of waiting time when compared with SSTF. The sacrifice made for this improvement is an increase in mean waiting time. At low arrival rate, SCAN behaves like FCFS/SSTF because the queue tends to be empty most of the time. For very high arrival rates, the area behind read/write head tends to become quickly repopulated and thus SCAN is able to reduce seek time considerably compared to FCFS. This algorithm is also known as LOOK/ELEVATOR in literature [16].

- **Circular Polling (C-Polling)**: C-Polling algorithm is a variant of Polling algorithm. In this algorithm, the read/write head serves requests in only one direction. To serve a queue of requests using this algorithm, the read/write head serves requests as in Polling algorithm but on reaching outer end of the disk, head moves directly back to the inner most cylinder without servicing any request on return path [5].

- **Circular SCAN (C-SCAN)**: This algorithm is a variant of SCAN algorithm. Disk arm provides service in only one direction as in Circular Polling. Read/write head serves requests as in SCAN algorithm and when there are no more pending requests in the current direction, it moves directly to the innermost cylinder having a request. This algorithm is also known as Circular LOOK (C-LOOK) in literature [16].

- **N-STEP SCAN**: In this algorithm, requests are grouped in sizes of N or less; that is minimum of (a) N, and (b) the current queue length L whenever the previous group has been completely serviced. Requests from a group are serviced as follows: (a) First scan in the direction for which the farthest request distance from current read/write head position is a minimum. (b) Scan back until the remaining requests have been served. This algorithm decreases the variance of individual waiting times while not significantly degrading throughput [20]. Lower variance of waiting time results from the fact that once a group has been formed, no reordering of requests is allowed. Under heavy loading conditions, this becomes a pure SCAN from one extreme cylinder with a request to the other extreme cylinder with a request.

- **ESCHENBACH Scheme**: The Eschenbach scheme [21] is a deterministic scanning algorithm designed for systems that normally operate under heavy loading conditions such as message switching system. An order E Eschenbach scan is defined as E revolutions per cylinder, with m/E sectors serviced per revolution (m = total number of sectors per track). In an order E scan, E − 1 sectors are skipped for each sector serviced. It has the following properties:

  (a) in order to allow the possibility of servicing all m sectors for a cylinder, the read/write head must remain in position for at least E revolutions; (b) if cylinders are labeled 0, 1, ..., C−1, the arm may spend E revolutions on cylinder 0 before moving to cylinder 1, E revolutions on cylinder 1, and so on until E revolutions are spent on cylinder N. The arm then moves directly to cylinder 0 without stopping at any of the intermediate cylinders.

- **VSCAN(R)**: For each real R in [0,1], VSCAN(R) algorithm [6] behaves as follows. VSCAN(R) maintains a current direction of read/write head (in or out) and services next that request with the smallest effective distance. The effective distance to a request in the current direction is simply its physical distance in cylinders from the current read/write head position. The effective distance to a request in the opposite direction is its physical distance plus R × (the total number of cylinders on the disk). Thus VSCAN(0.0) = SSTF and VSCAN(1.0) = SCAN.

  Coffman [4] has suggested a modification of VSCAN(R) that is most easily described in terms of implementation of VSCAN(R). Specifically, two queues of requests are maintained, one for cylinder numbers higher than the current position of the read/write head, the other for lower cylinder numbers. Each queue is sorted by distance from the current position of the read/write head. When a new request arrives, it is inserted at the appropriate position in the appropriate queue. When a new request is to be serviced, the effective distance to the first request in each queue is computed and that with the smaller effective distance is selected. Note only first request in each queue is considered, and no dynamic queue reshuffling is ever needed.

- **CVSCAN(N,R)**: This algorithm computes the effective distance by the method of VSCAN(R) but uses the average effective distance of N requests in each direction as the basis for deciding which of the two requests (physically nearest in each direction) is to be serviced next [10]. If there are fewer than N requests in one or both directions, say n_1 in one direction and n_2 in other, then use the average of min$\{n_1, n_2\}$ requests in each direction. As N increases, CVSCAN(N,0.2) yields a decrease in waiting time variance without a comparable sacrifice in the mean waiting time. However, results from [10] show that the improvement does not continue monotonically; CVSCAN(8,0.2) yielded both higher mean waiting time and higher standard deviation of waiting time than CVSCAN(5,0.2).

- **System V Standard (SVS)**: It is a quasi-SSTF sorting scheme applied within request batches of size N, with FCFS among batches [10].

  Within batches of size N the following algorithm is used. The first arrival to an empty queue becomes the request in service, but it retains its position at the head of the queue until its service is complete. For each request arrival thereafter, the algorithm proceeds as follows. Starting with entry n = 1 in the queue, it compares two distances:

  (1) Distance from arriving request to the entry n.

  (2) Distance from entry n to entry n+1 (taken to be infinity if entry n+1 does not exist).

  If first distance is the smaller of the two, the new request is inserted immediately after entry n. Otherwise n is incremented and the procedure is repeated, that is, two new distances are computed and compared. Requests are serviced from this single queue in order. The algorithm described above is applied only within successive groups of requests of fixed size N, with FCFS among groups. Xelos R01 (*Concurrent Computer Corporation's port of System V to their 3230 architecture*) uses **N=8** and UNIX System V Release 2.0 (AT & T 3B2/300) uses **N = 16**.

Chapter 3

MODEL AND METHODOLOGY

3.1 Introduction

Since mathematical analysis of disk head scheduling is hard, we developed an *open* and a *closed simulation model* for a disk to serve as the foundation for our study of disk scheduling algorithms. A simulator for the open and closed models was implemented in C language.

In this chapter, we will describe the open and closed models of a disk, the performance metrics used and some important implementation issues in our studies.

3.2 Assumptions made in our Simulation Studies

The following are the list of assumptions made in our studies. These assumptions are applicable to both open and closed models.

1. We assume a multi-user computing environment.
2. Only one disk module with a single read/write head is considered.
3. Tracks on the disk are referred to as *cylinders* throughout our study and disk under consideration has *NumCylinders* cylinders.
4. All jobs submitted by a user are I/O bound.

5. Time required by a request on CPU is negligible compared to the time required by it for I/O processing.

6. Arrival of I/O requests to disk is modeled as a Poisson process.

7. Cylinder numbers to which requests are made have uniform distribution between 0 and (NumCylinders−1).

8. Seek times are computed from a linear approximation of IBM 2314 seek time characteristic [20] and has the following form:
   Seek time = $\alpha + \beta \times$ (Seek distance in cylinders) for distances ≥ 1. The values of α & β parameters used in our study are slightly old but since these values are used by others, we kept them for consistency and comparing our results with that of others.

9. Rotation time plus transfer time is assumed to be constant and are lumped into $LatencyTime$ parameter.

10. No distinction is made between READ and WRITE requests, and the overhead to decide which request to service next is assumed to have no effect on the efficiency of the scheduling policy.

3.3 Performance metrics

Various performance metrics used in our study of some of the existing disk scheduling algorithms are as follows:

1. *Average response time for a cylinder*:
   For any cylinder i, where $0 \leq i \leq (NumCylinders - 1)$, average response time is the sum of response times of all requests for cylinder i divided by the number of requests for it.

2. *Overall average response time*:

   Sum of average response time for all cylinders divided by the number of cylinders.

3. *Fairness across cylinders*:

   Variance of average response time on per cylinder basis, that is,
   $fairness = \frac{\sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(RT_i - ORT)^2}}{ORT}$;
   where N = NumCylinders, RT_i is the average response time for cylinder i and ORT is the overall average response time.

3.4 Open Model of a Disk

3.4.1 Introduction

A disk is viewed as consisting of *NumCylinders* resources (Cylinders) and requests for these resources come in Poisson fashion with an average arrival rate of *ArrivalRate* requests per second. These requests are buffered in a single queue. Disk has one server (read/write head) providing service to these requests as shown in the figure 3.1. After service, a

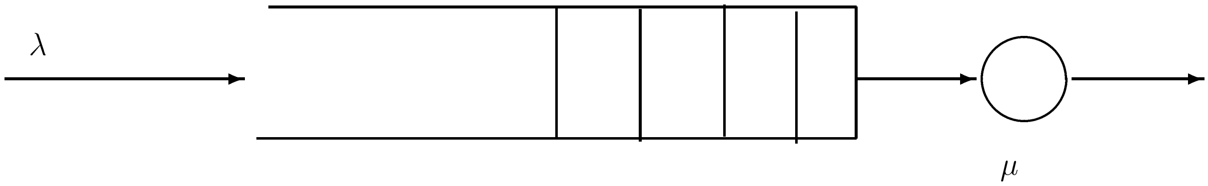

Figure 3.1: Open System Model of a Disk

request immediately leaves the system. The following subsections describe the simulation parameters, the process of generating workload and some of the important implementation issues for FCFS, SSTF, Polling, C-Polling, C-SCAN and SVS scheduling algorithms for open simulation model.

3.4.2 Simulation Parameters

Various parameters of the simulation in open model are shown in table 3.1.

Parameter	Meaning
ArrivalRate	Request arrival rate in requests per second
NumRequests	Number of requests
NumCylinders	Number of cylinders on disk
α	Startup time for head to start moving
β	Multiplier of seek distance
LatencyTime	Rotation time plus transfer time
BatchSize	Batch size used in SVS scheduling algorithm

Table 3.1: Open Model Simulation Parameters.

3.4.3 Workload Generation Process in Open Model

A request for disk service in open model is composed of two parts:

1. Request arrival time having exponential distribution, and

2. Cylinder number having uniform distribution.

Cylinder numbers are generated using following formula:
Cylinder_number $= rand() mod NumCylinders$, where NumCylinders is the number of cylinders on disk and rand() is a library function that returns non-negative uniformly distributed pseudo-random numbers.

Request arrival time is generated from the following formula:

Request_arrival_time $= -MEAN \times log(1.0 - drand48())$, where MEAN is the mean of exponential inter-arrival time distribution and is given by MEAN $= \frac{1000}{ArrivalRate}$ milliseconds. drand48() is a library function that returns successive pseudo-random numbers uniformly distributed between 0 and 1.

3.4.4 Implementation of Some of the Existing Disk Scheduling Algorithms

For each ArrivalRate, we generate NumRequests number of requests and write them to a file. Requests from this file are read into a queue which is then served by using FCFS, SSTF, Polling, C-Polling, C-SCAN and SVS disk scheduling algorithms. For the ease of implementation, all algorithms except FCFS and SVS have requests organised in the form of a table. The table is an array of *NumCylinders* elements, each element of which contains two pointers; (a) First pointer in table[i], for $0 \leq i \leq (NumCylinders - 1)$ contains a pointer to queue of requests for cylinder i. (b) Second pointer in table[i] is a pointer to the last request in the queue for cylinder i. This last request pointer improves the speed of adding a new request for cylinder i to the corresponding request queue. Requests in these queues and tables are then served using the corresponding algorithms.

In our studies, we have modified Polling, Circular Polling and Circular SCAN algorithms to *serve only one request at a time for each cylinder* because if all the requests for the current cylinder are serviced before moving onto the next cylinder, then at high arrival rates, the requests for other cylinders will have more waiting time as new requests will keep coming when service of requests for the current cylinder is in progress. This modification results in an increase in response time but variance of response time decreases. Hence from now onward, when we refer to Polling, C-Polling and C-SCAN, we mean modified Polling, C-Polling and C-SCAN which serve only one request for a cylinder at a time unless stated otherwise. Termination condition for each of these algorithms is that it should have served NumRequests number of requests. When NumRequests requests have been serviced by an algorithm, we measure the performance metrics mentioned in section 3.3.

- *First Come First Serve(FCFS)*: It reads newly arriving requests generated above into a single queue which is then served using FCFS algorithm. In addition to the performance metrics mentioned in subsection 3.3, we also measure *average queue length* for this algorithm.

- *Shortest Seek Time First (SSTF)*: Newly arriving requests for cylinder i are added to queue pointed to by table[i]. We thought of two variants of SSTF that one can try. These are as follows:

  – *Static SSTF(S-SSTF)*: To select a request for service, S-SSTF looks at the requests presently in the queue and chooses the one nearest to the current head position. Once a request is chosen, S-SSTF goes about servicing it without considering any new arriving request(i.e. S-SSTF doesn't change its decision till the chosen request has been serviced).

- *Dynamic SSTF(D-SSTF)*: D-SSTF selects a request for service as in S-SSTF case. But, while going about servicing that request, D-SSTF also considers the newly arriving requests, that is, when head has started moving towards the chosen request, if a new request arrives for a cylinder that is closer to the present position of head than the cylinder whose request put the head in motion, then D-SSTF changes its decision and starts head motion towards the newly arrived nearest request. This on-the-fly changing of decision of which request to serve next makes SSTF dynamic.

We have considered only Static SSTF in our studies. At high arrival rates, we expect D-SSTF will give better overall average response time than S-SSTF because of dynamic changing of decision of which request to serve next, the newly arriving request will have very very small response time. However, we expect that the discrimination against the extreme cylinders will be more in D-SSTF case than in S-SS3/csase. At low arrival rate, we expect S-SSTF and D-SSTF will behave in the same fashion. As far as implementation of S-SSTF is concerned, to select the next request for service, we search the table in both forward and backward direction to find a request nearest to the current head position. This request is the one to be serviced next. If we find two requests at equal distance from the current head position, we break the tie by choosing the one which arrived earlier into the system because of fairness considerations.

- *Polling*: Newly arriving requests for cylinder i are added to queue pointed to by table[i]. Requests from table are served using Polling algorithm. When there is no request in the system, the read/write head stays at the cylinder where it served a request last. When read/write head reaches either end of disk surface, it incurs a turn back cost of α milliseconds and starts moving in the reverse direction.

- *Circular Polling (C-Polling)*: Newly arriving requests are added to the table as in Polling algorithm and are serviced by using C-Polling algorithm. When read/write head reaches outer end of the disk, it incurs a turn back cost of α milliseconds and moves directly to the inner most cylinder without servicing any request on the return path. When there is no request in the system, the read/write head stays at the cylinder where it served a request last.

- *Circular SCAN (C-SCAN)*: We considered C-SCAN algorithm instead of SCAN because C-SCAN is a fairer version of SCAN [16]. Requests are added to table as above. When there is no pending request in the current direction, we search the queue associated with each cylinder starting from inner most cylinder. When a queue associated with lowest numbered cylinder having atleast one request is found, read/write head moves directly to that cylinder. The first request in this queue is the one to be serviced next.

- *System V Standard (SVS)*: BatchSize number of requests from newly arriving ones are added to svs queue using the algorithm described in the last chapter. The requests from svs queue are then serviced by using SVS algorithm.

3.5 Closed Model of a Disk

3.5.1 Introduction

This model consists of *NumTerminals* number of users accessing their files and data stored on a disk. A user submits a disk request and then waits till the request is serviced. After getting response to an earlier request , a user thinks for some time to analyse the results received before submitting next request. The requests wait in a single queue before service. Server is a single read/write head that services the requests in the queue. After a request has been serviced, it goes back to the terminal from which it came as shown in figure 3.2.

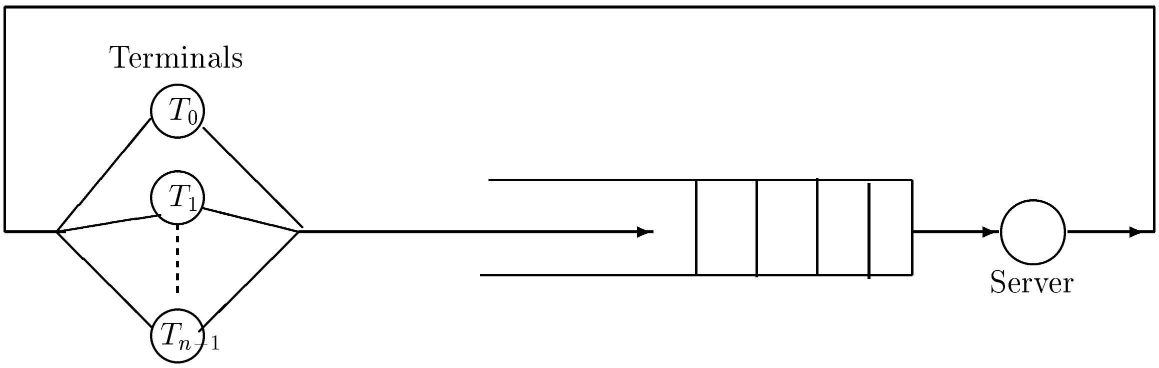

Figure 3.2: Closed System Model of a Disk

The following subsections cover closed model simulation parameters, workload generation process and some of the important implementation issues for same set of disk arm scheduling algorithms as in open model.

3.5.2 Additional Assumptions Applicable to Closed Model

In addition to the assumptions made in section 3.2, the following additional assumptions are also applicable to the closed model:

1. Each user submits a new job only after his/her previous job has been completed. When a user receives response to his/her earlier job, he/she thinks independently for some random amount of time before submitting next job. We have considered three distributions for think time: (1) Exponential (2) Uniform and (3) Deterministic. For each value of *Multiplier* parameter, the mean of each distribution is kept constant.

2. Cylinder numbers in a user's area to which he/she makes requests have uniform distribution between *lower* and *upper*, where *lower* and *upper* is the lowest and highest numbered cylinder in his/her area respectively.

3.5.3 Simulation Parameters

In the closed model, various parameters of the simulation alongwith their meaning are listed in table 3.2.

Parameter	Meaning
NumRequests	Number of requests
NumCylinders	Number of cylinders on disk
NumTerminals	Number of terminals or users accessing disk
RANGE	Number of cylinders allocated to each user
ThinkTimeDistn	Think time distribution
Multiplier	Mean of think time distribution is Multiplier times the expected service time.
α	Startup time for head to start moving
β	Multiplier of seek distance
LatencyTime	Rotation time plus data transfer time
BatchSize	Batch size used in SVS scheduling algorithm

Table 3.2: Closed Model Simulation Parameters.

3.5.4 Workload Generation Process in Closed Model

A request in our closed model consists of *number of the terminal* issuing request, *cylinder number* to which request is made and *time* at which this request is issued. Request also has one more field, named *request out time*, for recording time when a request completes service. This field helps in generating next request from the same terminal after current request has been serviced. Here we require *NumTerminals* number of independent streams for generating cylinder numbers and also same number of independent streams to generate request arrival times because each user makes requests to cylinders in his/her own part of disk and thinks independently of any other user. We define two 2-dimensional arrays as

```
unsigned short xsubi[NumTerminals][3];
unsigned short ysubi[NumTerminals][3];
```

where xsubi[i][3] and ysubi[i][3] contain initial values for nrand48() and erand48() library functions used to generate cylinder numbers and request arrival time for requests from terminal i, where i varies between 0 and (NumTerminals − 1). If we specify different initial values in three element arrays xsubi[i][3] and ysubi[i][3] for $0 \leq i \leq (NumTerminals - 1)$, we get uniformly distributed pseudo-random numbers from different independent streams.

Cylinder numbers are generated as follows:

```
lower = RANGE * terminalNum
Cylinder_number = lower + nrand48(xsubi[terminalNum]) mod RANGE,
```

where RANGE is the number of cylinders allocated to the user issuing requests from terminal terminalNum and nrand48(xsubi[terminalNum]) is a library function that returns positive integers uniformly distributed over the range 0 to $2^{31} - 1$.

For request arrival time generation, we first compute the mean of think time distribution as follows:

Mean distance $E[d]$ that head needs to travel before servicing a request which arrived into an empty system is given by a well known classical result as:

$E[d] = \frac{N}{3}$, where N = Number of cylinders on the disk

Therefore, Expected seek time = $\alpha + \beta E[d] for E[d] \geq 1$,

Expected service time $E[Ts]$ = Expected seek time + LatencyTime, and

Mean of think time distribution MEAN = Multiplier $\times E[Ts]$.

We considered three distributions for think time and we generated request arrival time for each distribution as follows:

- *Exponential think time distribution*: Here we use the following expressions for request arrival time generation:

  ```
  thinktime = -MEAN * log(1.0 - erand48(ysubi[terminalNum]);
  reqptr->req.reqintime = reqptr->req.reqouttime + thinktime;
    Note: Initially reqptr->req.reqouttime = 0;
  ```

  where *MEAN* is the mean of exponential think time distribution, terminalNum is the number of the terminal issuing request and erand48(ysubi[terminalNum]) is a library function that returns uniformly distributed successive pseudo-random numbers between 0 and 1.

- *Uniform think time distribution*: Here we consider the range of uniform distribution to be between $0.5 \times MEAN$ to $1.5 \times MEAN$. Request arrival time is generated as follows:

```
scaleFactor = 0.5 + erand48(ysubi[terminalNum]);
thinktime = scaleFactor * MEAN;
reqptr->req.reqintime = reqptr->req.reqouttime + thinktime;
```

- *Deterministic think time distribution*: Here think time is constant and is equal to MEAN. Initially, we use a random number generator to generate first request from each terminal. After a request has been serviced, the next request from the terminal that issued this request is generated as follows:

```
thinktime = MEAN;
reqptr->req.reqintime = reqptr->req.reqouttime + thinktime;
```

Requests from all terminals are added to a single queue pointed to by a header. For the ease of implementation, all algorithms except FCFS and SVS have requests organised in the form of a table. A request for cylinder i from header queue is added to queue pointed to by table[i]. Therefore, header queue here plays the same role as was played by a file containing requests in open model.

3.5.5 Implementation of the same set of Disk Scheduling Algorithms as in Open Model

For each value of Multiplier parameter, we read requests from terminals and add them to the header queue. FCFS serves requests from this queue in order, Shortest Seek Time First (SSTF), Polling, C-Polling and C-SCAN algorithms read requests from header queue into the corresponding tables and serve them whereas SVS algorithm adds BatchSize number of requests from header queue to svs_queue and serves them. When a request has been serviced, we record its service completion time in the unfilled *request out time* field. The next request from the terminal whose request just completed service comes at time *request out time plus think time*. We keep reading requests from terminals and servicing them till NumRequests have been serviced, at which time we measure the performance metrics mentioned in section 3.3.

Chapter 4

OPEN MODEL EXPERIMENTAL RESULTS

4.1 Introduction

Having presented in last chapter the performance metrics used in our study and some of the implementation issues for FCFS, SSTF, SVS, C-SCAN, Polling & C-Polling disk scheduling algorithms, in this chapter, we present our open model simulation results. Our results match qualitatively with the ones obtained in [20] where the performance metrics were expected seek time, expected service time for individual requests and squared coefficient of variation of waiting time. Note we have not measured the *confidence intervals* in our studies and also there are no supporting numbers for our simulation results.

4.2 Open Model Simulation Parameter Values

The simulation parameters mentioned in table 3.1 alongwith their corresponding values used in our study are shown in table 4.1.

Parameter	Value(s)
ArrivalRate	1.0, 3.0, 5.0, 10.0, 15.0, 20.0 requests per second
NumCylinders	1000
NumRequests	1000000
α	30.0 milliseconds
β	0.50 milliseconds/cylinder
LatencyTime	8.0 milliseconds
BatchSize	16

Table 4.1: Open Model Simulation Parameter Values

4.3 Simulation Results

In this section, we present open model simulation results in the form of graphs for the set of algorithms considered in last chapter and analyse their behaviour for various arrival rates. For the description below, refer graphs 4.3.1 to 4.3.3.
At low arrival rates, Shortest Seek Time First (SSTF), Circular SCAN(C-SCAN) and System V Standard(SVS) algorithms behave in the same fashion as First Come First Serve(FCFS) algorithm because number of requests in the system is very low and these algorithms have little choice to make from. Higher average response time of Polling and Circular Polling algorithms at low arrival rates is due to the fact that these algorithms do not change the direction of head movement even when there is no pending request in the current direction. The requirement to make head goto the extreme end adds to the waiting time of request(s) at other extreme end and hence higher average response time.

As the arrival rate increases, FCFS is the first algorithm to saturate. By saturation, we mean that arrival rate becomes more than service rate and thus the number of outstanding requests in the system becomes more. The reason for FCFS to saturate before any of the other algorithms is that it does not perform any optimization. We observed in our experiment that around an arrival rate of 5.0 requests per second, FCFS saturated whereas in Teorey & Pinkerton's [20] experiment for 200 cylinders, FCFS saturated at an arrival rate of 10.0 requests per second. Saturation at low arrival rate makes FCFS undesirable under most operating conditions.

Shortest Seek Time First(SSTF) has the lowest average response time for all arrival rates but the variance in average response time is much more than in any other algorithm. This is because as the arrival rate increases, SSTF can most of the time find a request near the position where it served a request last and thus read/write head tends to be confined to a part of disk for some time, thereby discriminating against other waiting requests. Large average response time for the requests to the extreme end cylinders compared to the ones in the middle of disk is due to the *"near-sightedness"* property of SSTF algorithm [5].

System V Standard (SVS) algorithm is the next algorithm to saturate after FCFS because it uses FCFS between batches and quasi-SSTF within a batch of requests. We found that SVS saturated around an arrival rate of 15.0 requests per second with SVS queue batch size of 16 requests. At low arrival rates, we observed that SSTF and SVS average response time versus cylinder number curves are very close to each other. This is because at low arrival rates, the number of requests waiting in request batch other than the one currently in service is very less and since SVS uses SSTF kind of algorithm within a request batch, it behaves like SSTF. Small improvement in average response time curve for SSTF over SVS is because SSTF looks at all the requests currently in the system to choose a request nearest to present head position to be serviced next but SVS looks only at requests within the batch currently in service to select the one to be serviced next.

In the curve for Polling algorithm, we found that the cylinders in the middle of disk have lower average response time compared to the ones at the extreme ends. This is because two full sweeps of the entire disk occur between the servicing of two consecutive requests at an extreme end cylinder whereas only one sweep of disk occurs between servicing of two consecutive requests at cylinders in the middle of disk.

In Circular Polling (C-Polling) algorithm, the average response time versus cylinder number curves for low arrival rates have large average response time for cylinders near the inner end of the disk compared to the ones near the outer end. This is because when number of requests in the system is less, the requirement to make read/write head change direction only on reaching outer end causes a delay in servicing of requests for cylinders near the inner end. The *full stroke time* i.e. the time reqmakrd to move read/write head from outer end to the inner end causes further delay in the servicing of the requests waiting at the inner end. At higher arrival rates, middle cylinders and outer end cylinders have almost same average response time because requests for these cylinders wait for nearly same amount of time before service. The cylinders near the inner end of disk have higher average response time compared to the middle and outer end cylinders. This is due to the delay introduced by full stroke time required to move read/write head from outer end to the inner end during which no request is serviced. As the arrival rate increases, fewer and fewer cylinders near inner end tend to have average response time more than the rest of the cylinders because at high arrival rate, it is not only the requests for inner end cylinders that wait for service during full stroke time but also the requests for other cylinders.

At low arrival rate, average response time versus cylinder number curve for Circular SCAN (C-SCAN) algorithm has shape similar to that of FCFS because of very small number of requests in the system, C-SCAN cannot perform any seek optimization. As the arrival rate increases, the shape of C-SCAN curve becomes similar to that of C-Polling curve but the number of cylinders near the inner end of disk having more response time than rest of the cylinders is less in C-SCAN case. This occurs because C-SCAN does not make read/write head to goto the extreme ends of disk even if there are no pending requests there. Hence the requests near the inner end of the disk wait for lesser amount of time before getting service in C-SCAN case than in C-Polling case.

4.4 Non-Stop Circular Polling (NSCP)

We have seen from the above discussion as well as from the experimental results for the existing disk scheduling algorithms which we've considered, that these algorithms discriminate against the requests for extreme end cylinders. Average response time versus cylinder number curves for these algorithms deviate considerably from the one shown in figure 1.2. We developed a *Non-Stop Circular Polling* algorithm which is a variant of the original Circular Polling (C-Polling) algorithm. This algorithm operates like the original

C-Polling algorithm with two modifications, namely (a) It does not serve bulk requests for a cylinder. Only one request for a cylinder is served at a time. (b) The read/write head continuously keeps moving from one end of the disk to the other end even when there is no request in the system. NSCP algorithm approximates the curve shown in figure 1.2 reasonably well and hence is close to absolute fairness. NSCP algorithm gives following performance improvements over original C-Polling algorithm.

- Better average response time than the original C-Polling algorithm with above mentioned modification (a) added.
- Variance of average response time is less than that in case of C-Polling for all arrival rates we experimented.

Reasons for the above improvements are the following. At low arrival rate, consider a situation where there is only one request currently in the system. After servicing this request, the head will stop at its present position in case of C-Polling algorithm whereas it will keep moving in the current direction in NSCP case. Now, if the next request comes at a distance d, for $2 \leq d \leq (NumCylinders - 1)$, from the current position of C-Polling head, then chances of finding NSCP head somewhere near the new request are more than C-Polling head because NSCP head has covered some of the distance d during idle period whereas C-Polling head will start moving only when the new request enters the system. Thus the amount of time the new request waits in the system is comparatively less than that in C-Polling case. This reduction in waiting time contributes directly to the improvement in response time. However, if NSCP head had crossed the cylinder for which the new request came, new request has to wait more in NSCP case than in C-Polling case but NSCP head could be nearing some other request in the system. Thus, if NSCP is increasing the waiting time of some requests in one hand, it is also reducing the waiting time of other requests in other hand. So, on average, NSCP performs better than C-Polling. We expect NSCP to behave in the same fashion as C-Polling at very high arrival rates because system will hardly be empty and so C-Polling head will not stop too.

One possibility where NSCP can perform poorly than C-Polling is when a new request always comes to a cylinder between current C-Polling head position and current

NSCP head position. But, in a typical computer system, requests to a disk are uniformly distributed across all its cylinders, therefore the chances of this happening are very low.

The *overall average response time versus fairness* at ArrivalRate and *overall average response time versus overall coeff. of variance of response time* at ArrivalRate for each algorithm are shown in graphs 4.3.4 and 4.3.5 respectively. These curves show how fair an algorithm is at a particular arrival rate and how each algorithm changes its behaviour as arrival rate changes. The curve showing trade off between *overall average response time and fairness* at constant arrival rate is shown in graph 4.3.6. This curve clearly shows the amount one has to pay in terms of average response time as he tries to be more fair.

Chapter 5

CLOSED MODEL EXPERIMENTAL RESULTS

5.1 Introduction

In the previous chapter, we studied the performance of some of the existing disk scheduling algorithms using an open model. Also we compared these results with the results of our Non-Stop Circular Polling algorithm and we found that our algorithm approximated the curve shown in figure 1.2 reasonably well. In this chapter, we present the closed model experimental results for the same set of algorithms as in open model. Here also, we have not measured the confidence intervals and there are no supporting numbers for our simulation results.

5.2 Closed Model Simulation Parameter Values

The values of simulation parameters for closed model listed in table 3.2 are shown in the table 5.1.

Parameter	Value(s)
NumTerminals	200
NumCylinders	1000
NumRequests	1000000
RANGE	5 Cylinders per terminal
Multiplier	10, 100 & 500
ThinkTimeDistn	Exponential, Uniform & Deterministic
α	30.0 milliseconds
β	0.50 milliseconds/cylinder
LatencyTime	8.0 milliseconds
BatchSize	16

Table 5.1: Closed Model Simulation Parameter Values

5.3 Simulation Results

We used the parameters listed in table 5.1 in our closed model experiment for FCFS, SSTF, Polling, C-Polling, C-SCAN, SVS and NSCP algorithms to study their behaviour. In this section, we present closed model simulation results. For the discussion below, refer graphs 5.3.1 to 5.3.6.

When value of Multiplier parameter is high(i.e. Multiplier = 500), think time is also high because mean of think time is directly proportional to Multiplier parameter. Large value of think time corresponds to very low arrival rate because users will be spending most of their time in thinking mode. Low arrival rate implies that number of requests waiting in the queue for service is very less. Hence FCFS, SSTF, SVS and C-SCAN algorithms behave in the same fashion. The average response time versus cylinder number curves for these algorithms are very close to each other. However Polling, C-Polling and NSCP algorithms have slightly more average response time because these algorithms reverse the direction of head movement only on reaching extreme ends. This increases the waiting time of requests at other extreme end and hence have more response time. This indeed was the case in our open model too when arrival rate was very low. Also, for exponential & uniform think time distributions, the average response time versus cylinder number curves for all these algorithms are very similar. This is because mean of both exponential

& uniform think time distributions is kept same and also since we are taking average of the response times of almost 1000 requests for each cylinder, the average response time for each cylinder will be nearly same in both exponential & uniform distributions even though the variation in response time in the exponential case can range from 0 to infinity.

If the value of Multipler parameter is small(i.e. Multiplier = 10), then mean of think time distribution is small because of direct proportionality between them. Small think time implies that users will be spending very little amount of time in thinking mode and would be submitting requests most of the time, thereby making arrival rate high. For large arrival rate, FCFS algorithm has the worst average response time for deterministic, uniform & exponential think time distributions. This is because FCFS does not perform any optimization on seek time and large seek time results in large response time. The average response time versus cylinder number curves are flat for all algorithms except Polling and SSTF.

For uniform and exponential distributions, SSTF and Polling algorithms have better response time in the middle than at the ends. As was mentioned in last chapter for Polling algorithm, two full sweeps of entire disk surface occur between the servicing of two requests for an extreme cylinder whereas only one sweep occurs between servicing of two requests for a cylinder near the middle of disk. The two full sweep time makes the waiting time of requests for extreme cylinders more than that of requests for middle cylinders. Therefore middle cylinders have better average response time compared to the extreme cylinders. As far as SSTF is concerned, since it always looks for requests closest to the current head position, the requests for extreme cylinders suffer considerably because when arrival rate is high, it can find most of the time a request near the cylinder where it served a request last. Thus head tends to be confined to the middle of disk most of the time, hence better response time for middle cylinders than that for extreme cylinders.

Because of feedback from output to the input and circular servicing of requests by C-SCAN, C-Polling and NSCP algorithms, on an average, the amount of time requests for each cylinder wait in queue before service is almost same, thereby making average response time for all cylinders to be approximately same and hence the curves for these algorithms are straight.

For SVS algorithm, consider any batch B_i of *BatchSize* requests. Let T_i be subset of terminals whose requests are in the batch B_i. The requests at the front of this batch have small waiting time because SVS algorithm serves requests from a batch in order. After batch B_i has been serviced, next set of requests from terminals T_i will not be in same batch because number of terminals is not a multiple of BatchSize and also because of exponential & uniform think time distributions. So, next requests from terminals whose previous requests in batch B_i waited least before service will have to wait more in the next batch in which they join because these requests will be joining a partially complete batch. Therefore, the terminals whose requests in the current batch have very low response time, the next requests from these terminals will have high response time when the batch in which these requests join is serviced. Thus on average, the response time of all the requests from terminals is same and hence average response time curve is a straight line. Since SVS algorithm uses FCFS among batches, the quasi-SSTF algorithm used to add requests to a batch results in some amount of seek time optimization. This seek time optimization results in reducing the average response time and hence the average response time versus cylinder number curve for SVS is below the one for FCFS algorithm.

For deterministic think time distribution, average response time versus cylinder number curve for Polling algorithm has shape like a *step function*. The Polling algorithm serves the requests in the middle of disk approximately two times faster than those at the extreme ends. Suppose that head is moving from inner end to the outer end, serving requests for cylinders as it passes by. After a request has been serviced, the terminal from which this request came immediately enters thinking mode. After head has put all the terminals in the thinking mode during this pass from inner end to the outer end, the first terminal T_i, $0 \leq i \leq (NumTerminals - 1)$, out of thinking mode which the head will encounter when it takes a turn back from outer end is the one such that the time taken by head to travel from a cylinder(whose request was serviced during inner end to outer end pass of head) in the RANGE of T_i to another cylinder in the RANGE of T_i is atleast equal to the think time. That is, $[(\alpha + \beta \times RANGE + LatencyTime) \times T_i + \alpha] \geq ThinkTime$.

For Multiplier = 10.0, the ThinkTime = 2045.00 milliseconds. By substituting the values of other parameters from table 5.1, we get $T_i = 50$ from outer end. This implies that 150^{th} terminal will have a cylinder in its RANGE whose request will have response time close to zero. After servicing request from this terminal, the head will always find a request waiting to be serviced till it reaches the inner end of the disk because think time of the terminals between 150^{th} and 1^{st} terminal is over. The response time of the requests increases as head moves after servicing request from 150^{th} terminal because these requests had come earlier and waited longer. Since RANGE = 5 Cylinders per terminal, approximately around 750^{th} Cylinder will have very low average response time and hence the upper discontinuity.

Because of symmetry, when head turns back from inner end, 50^{th} terminal having 250^{th} cylinder in its RANGE will have very low average response time and hence the lower discontinuity. Taking average after a large number of head movements from inner end to outer end and from outer end to inner end, the average response time for the requests for cylinders in the RANGE of 50^{th} terminal to 150^{th} terminal will be same and hence the flat line between two discontinuities.

The overall average response time versus fairness at Multiplier parameter and overall average response time versus overall coeff. of variance of response time at Multiplier parameter for exponential, uniform and deterministic think time distributions for each algorithm are shown in the graphs 5.3.7 to 5.3.12. The curves showing trade off between overall average response time and fairness at constant Multiplier parameter for various think time distributions considered in closed model are shown graphs 5.3.13 to 5.3.15. These curves show the amount one has to pay in terms of average response time for fairness under various think time distributions.

Chapter 6

SUMMARY AND FUTURE RESEARCH

6.1 Summary of Results

Ever since people started moving away from batch proces3/cs systems to time sharing systems, their objective shifted from merely throughput consideration to both throughput and establishing fairness among a set of users competing for system resources. In this thesis, we restricted our attention to a single system resource, namely, a Disk. In a computer system, an ordinary user has not control over the placement of his data and programs on the disk. As a result, the users whose data and programs is placed on the middle of the disk get better treatment compared to the ones whose data is placed at the extreme ends of the disk because at medium to high arrival rates, disk read/write head tends to stay in the middle of the disk, thereby providing better service to these users. However, from a user's perspective, it does not matter where his files are kept, he will be concerned only if he is being discriminated against by the system in favor of other users who were lucky enough that their data got placed on the middle of disk. We developed an open and a closed model of a disk and simulated FCFS, SSTF, C-SCAN, Polling, C-Polling and SVS disk scheduling algorithms. We analysed how fair these algorithms are and how they change their behaviour as arrival rate changes. We also proposed Non-Stop Circular Polling disk scheduling algorithm that does not discriminate against the users who have been allocated a part of the disk near the extreme ends. We compared the results obtained from our algorithm with some of the existing algorithms and found our results to be quite satisfactory as far as fairness is concerned. To summarize, this thesis has shown that FCFS, SSTF, Polling, Circular Poling, C-SCAN and SVS are not the algorithms of choice if one is interested in fairness on per cylinder basis. Instead, we recommend the use of Non-Stop Circular Polling algorithm that has the fairness property at all arrival rates we considered.

6.2 Future Research Directions

We developed Non-Stop Circular Polling algorithm ignoring the time that an I/O activity takes on CPU and so the requests for disk service go directly to the disk system. An attractive avenue to explore would be to investigate how NSCP algorithm behaves on an actual system. Also, it would be nice to perform some optimization on NSCP so as to obtain fairness while not degrading average response time considerably. It would be interesting to build a general model of a computer system similar to [11], that combines one of the fair CPU scheduling algorithms such as *Round Robin Multilevel Queues with Feedback* and our NSCP algorithm into a single fair algorithm and study the performance impact on the system as a whole.

Bibliography

[1] Bach, Maurice J.,*"The design of the UNIX Operating System"*, *Prentice Hall of India Private Limited, New Delhi-110 001, 1989.*

[2] Bagrodia, Rajive L., and Tsay, Yih-Kuen, *"A Real-Time Algorithm for Fair Interprocess Synchronization"*, Computer Science Department, University of California, Los Angeles, Los Angeles, CA 90024.

[3] Coffman, Edward G., Jr. and Denning, Peter J., *Operating Systems Theory*, pp. 209-218, PHI Series in Automatic Computing.

[4] Coffman, E. G., Jr., and Hofri, Micha, *"On the Expected Performance of Scanning Disks"*, *SIAM J. Comput.,*, 11(1) pp. 60-70(Feb., 1982).

[5] Coffman, E. G., and Gilbert, E., *"Polling and Greedy Servers on a line"*, *AT & T Bell Labs.*

[6] Daniel, S. and Geist, Robert ,*"V-SCAN: An Adaptive Disk Scheduling Algorithm"*, *Proc. of the IEEE Int. Symp. on Comp Sys. Org., New Orleans*, March, 1983.

[7] Denning, Peter J., *"Effects of Scheduling on File Memory Operations"*, *Proc. AFIPS SJCC*, 31(1967), pp. 9-21.

[8] Frank, H.,*"Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems"*, *Journal of the ACM*, 16(4) pp. 602-620(Oct., 1969).

[9] Fuller, S. H., *"Minimal-Total-Processing Time Drum and Disk Scheduling Disciplines"*, *Comm. of the ACM*, 17(7) pp. 376-381.(July, 1974).

[10] Geist, Robert, Reynolds, Robert and Pittard, Eve *"Disk Scheduling in System V"*, Department of Computer Science, Clemson University, Clemson, South Carolina, USA 29634-1906.

[11] Gopalakrishnan, Ramamurthy, *"An Analytic Model for UNIX systems"*, *AT & T Technical Journal* Sept-Oct., 1988, pp. 86-100.

[12] Gotlieb, C. C., and MacEwen, G.H., *"Performance of Movable-Head Disk Storage Devices"*, *Journal of the ACM*, 20(4) pp. 604-623(Oct., 1973).

[13] Ganger, Gregory R. and Patt, Yale N., *"The Process-Flow Model: Examinig I/O Performance from the System's Point of View"*, Department of Electrical Engg. and Computer Science, University of Michigan, Ann Arbor 48109-2122.

[14] Henry, G. J., *"The UNIX System: The Fair Share Scheduler"*, *AT & T Bell Laboratories Technical Journal*, 63(8), Oct., 1984, pp. 261-273.

[15] Oney, C. Walter, *"Queueing Analysis of the Scan Policy for Moving-Head Disks"*, *Journal of the ACM*, 29(3) pp. 397-412 (July, 1975).

[16] Peterson, James L. and Silberschatz, Abraham, *"Operating System Concepts"*, Second Edition, *Addison-Wesley Publication Company.*

[17] Swartz, G. Boyd, *"Polling in a LOOP system"*, *Journal of the ACM*, 27(1) pp. 42-59(Jan., 1980).

[18] Tanenbaum, Andrew S., *"Operating Systems: Design and Implementation"*, *Prentice Hall of India Private Limited, New Delhi-110 001, 1990.*

[19] Trivedi Kishor S., *"Probability & Statistics with Reliability, Queueing, and Computer Science Applications"*, *Prentice Hall of India Private Limited, New Delhi-110 001, 1988.*

[20] Teorey, Toby J. and Pinkerton, Tad B., *"A comparative Analysis of Disk Scheduling Policies"*, *Comm. of the ACM*, 15(3), pp. 177-184(March, 1972).

[21] Weingarten, A., *"The Analytical Design of Real-Time Disk Systems"*, *Proc. IFIP Congr. 1968*, North Holland Pub. Co., Amsterdam, pp. D131-D137.

[22] Wilhelm, N., *"An Anomaly in Disk Scheduling: A Comparision of FCFS and SSTF Using an Emperical Model for Disk Accesses"*, *Comm. of the ACM*, 19(1) (Jan, 1976).

[23] Wilhelm, N., *"A General Model for the Performance of Disk Systems"*, *Journal of the ACM*, 24(1) pp. 14-31 (Jan, 1977).