A Disk Head Scheduling Simulator

Steven Robbins
Department of Computer Science
University of Texas at San Antonio
srobbins@cs.utsa.edu

Abstract

Disk head scheduling is a standard topic in undergraduate operating systems courses. Disk drives were once fairly simple devices with little intelligence. Disk head scheduling and bad block mapping were done exclusively by operating systems. As disk drives became larger and faster, they took on some of these tasks. Modern drives often have a large cache and hide their internal structure from the outside world. In spite of changes in disk technology, the teaching of disk head scheduling has changed little over the last decade. This paper describes a disk head scheduling simulator that allows students to explore traditional disk scheduling algorithms as well as the consequences of modern disk technology. The simulator, which is written in Java and is freely available, can be run remotely from a browser or downloaded for local execution. We present methods for modifying the traditional curriculum to make the presentation of disk head scheduling more relevant and interesting.

Categories & Subject Descriptors

K.3 [**Computers & Education**]: Computer & Information Science Education - Computer Science Education

General Terms

Disk scheduling

Keywords

disk, scheduling, logical block addressing, LBA, simulation

1 Introduction

Disk scheduling algorithms have been studied for many years [1] and are discussed in most undergraduate operating systems textbooks [2, 4, 5, 6]. The treatment of disk scheduling in these books has changed little in the last 10 years, even though there have been major changes in disk technology.

A standard treatment of disk scheduling is usually preceded by a discussion of traditional disk structure, describing the disk as a number of platters with a head for each surface. Each surface is divided into a fixed number of tracks, and each track has a fixed number of sectors. Each surface has one disk head, and the disk heads move in unison. At any given time, the heads can read from sectors on a given track from one surface. The collection of sectors available without moving the head is called a cylinder.

Disk drives have constant angular velocity. Unlike CDROMs, they always spin at the same rate, independent of the head position. As a result, the surface of the disk moves past the disk head faster when the head is over the outer tracks of the disk than when it is over the inner tracks. If each track has the same number of sectors, the sectors are physically bigger on the outer tracks. The physical properties of magnetic storage limit the number of sectors per track based on the bit density of the innermost tracks, resulting in inefficient use of the disk surface.

Modern disk drives improve storage capacity by putting more sectors per track on the outer tracks than on the inner ones. The disk geometry is made transparent to the operating system in one of two ways. One method is to present a traditional, but physically inaccurate, view of the disk to the operating system. This approach requires no change in the operating system software. The disk appears to have a given number of heads (surfaces), tracks per surface, and a fixed number of sectors per track. The numbers are chosen so that the product of these three numbers equals the total number of available sectors on the disk. The operating system uses this logical view to schedule accesses and to communicate with the disk controller. The disk controller converts the three numbers (logical head, logical track, logical sector) into three numbers corresponding to the physical head, physical track, and physical sector.

Another method of handling nonuniform disk geometry is Logical Block Addressing (LBA). With LBA, the operating system views the disk as an array of sectors with no indication of how they are laid out on the disk. The operating system must understand LBA and communicate with the disk controller using a logical block address instead of separate head, track and sector numbers.

This paper presents a brief discussion of the traditional treatment of disk scheduling in Section 2. Section 3 introduces an interactive simulator that allows students to experiment with different traditional algorithms. Section 4 describes additional uses of the simulator. The simulator is freely available and runs on any machine with a Java runtime system.

The performance impact on disk scheduling by an operating system using a logical view of the disk is discussed in Section 5. The effects of LBA on disk scheduling are discussed in Section 6. The transparent mapping of bad sectors simplifies disk management from the system administrator's point of view. The effects of bad blocks on disk scheduling are discussed in Section 7. Section 8 describes some student projects and the use of the simulator in the curriculum. Finally, Section 9 discusses some limitations of the simulator and its availability.

SIGCSE'04, March 3-7, 2004, Norfolk, Virginia, USA.

2 Traditional Disk Scheduling

Disk scheduling discussions are mainly concerned with seek time, the time it takes to move the heads from one cylinder to another. The rotational latency, the time needed to wait for the head to be over a particular sector, is usually not considered. The discussions assume that the operating system has the information needed to do the scheduling. That is, the operating system knows which physical cylinder is being accessed. This is a reasonable assumption when a disk has the traditional structure and all tracks have the same number of sectors. If the block number is known, a simple division gives the cylinder number.

Most operating systems textbooks treat a subset of the standard disk scheduling algorithms. For a description of these algorithms, see [7]. Operating systems textbooks typically discuss the following simplified disk scheduling problem: *Suppose you have a fixed list of cylinders that need to be accessed. In what order should the accesses take place?* Some texts mention rotational latency and transfer time, but these costs are typically ignored when disk head scheduling is described. The typical algorithms discussed include:

First-Come, First Served (FCFS) in which the seeks are done in the order requested.

Shortest Seek Time First (SSTF or SSF) in which the next seek is to the position closest to the current head position. Most books report that SSTF typically requires less total head movement than FCFS but that it can cause starvation when there are a sufficient number requests in a localized area of the disk.

SCAN and **LOOK** are sometimes called elevator algorithms. In these algorithms the disk head travels from one end of the disk to the other and back again. SCAN always goes to the extreme ends of the disk even if there are no requests there, while LOOK changes direction at the largest and smallest requested cylinder. Some books completely ignore the SCAN version or assume that the algorithm they call SCAN behaves like LOOK. The SCAN and LOOK algorithms spend more time at the center of the disk than at the ends.

C-SCAN and **C-LOOK** are variations in which requests are handled in only one direction. These algorithms tend to minimize the maximum delay experienced by new requests.

FSCAN, or frozen SCAN is like SCAN or LOOK but freezes the list of blocks to be scanned once scanning starts. New requests are put in a separate buffer. This is essentially a double buffering technique that gives preference to old requests over new ones.

3 The Disk Head Scheduling Simulator

The disk head simulator allows students to explore all of the algorithms described in Section 2. The simulator can be run either as a Java applet, remotely from a browser, or locally as a Java application. The results of the simulation can be displayed graphically or in tables.

In Silberschatz [4], the algorithms FCFS, SSTF, SCAN, C-SCAN and C-LOOK are compared assuming that the head starts at cylinder 53 and the queue of pending requests contains 98, 183, 37, 122, 14, 124, 65 and 67. We will refer to this as Experiment 1. Figure 1 shows diagrams created by the simulator similar to Figures 14.1 and 14.2 of the Silberschatz book for the FCFS and SSTF algorithms. Each horizontal line represents a scan of the disk in one direction, with tic marks at the positions of accessed blocks. Figure 2 shows various statistics calculated for these two runs. FCFS has a mean seek of 71.11 and SSTF has a mean seek of 26.22. Figure 3 shows the actual seeks done for the FCFS algorithm. Similar data can be produced for the examples given in other textbooks.

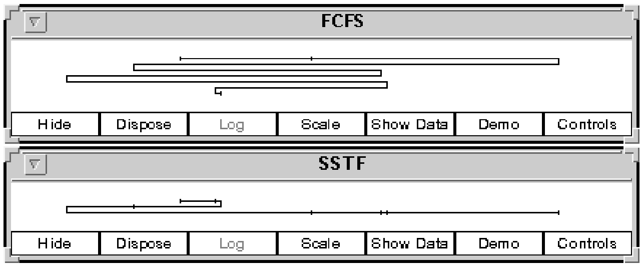
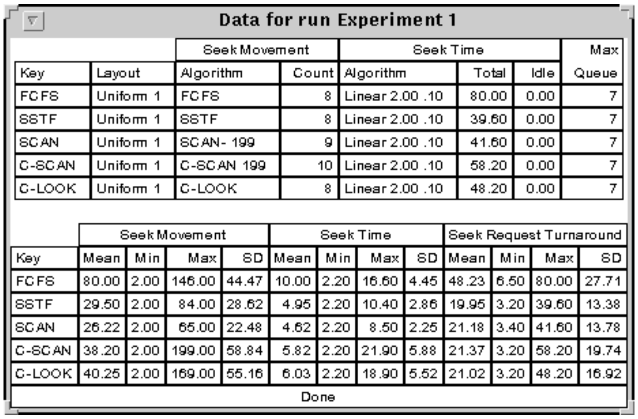


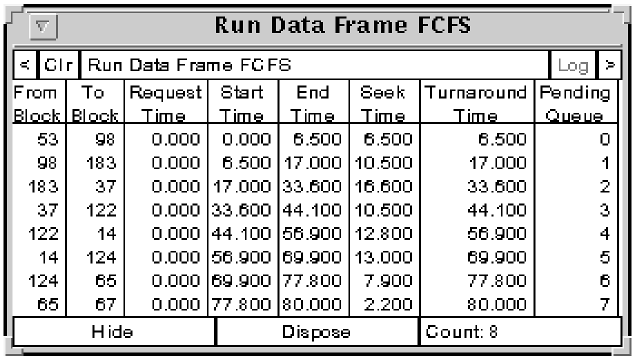
Figure 1: Two algorithms from Experiment 1.


Data for run Experiment 1

		Seek Movement		Seek Time			Max
Key	Layout	Algorithm	Count	Algorithm	Total	Idle	Queue
FCFS	Uniform 1	FCFS	8	Linear 2.00 .10	80.00	0.00	7
SSTF	Uniform 1	SSTF	8	Linear 2.00 .10	39.60	0.00	7
SCAN	Uniform 1	SCAN- 199	9	Linear 2.00 .10	41.60	0.00	7
C-SCAN	Uniform 1	C-SCAN 199	10	Linear 2.00 .10	58.20	0.00	7
C-LOOK	Uniform 1	C-LOOK	8	Linear 2.00 .10	48.20	0.00	7

	Seek Movement				Seek Time				Seek Request Turnaround			
Key	Mean	Min	Max	SD	Mean	Min	Max	SD	Mean	Min	Max	SD
FCFS	80.00	2.00	146.00	44.47	10.00	2.20	16.60	4.45	48.23	6.50	80.00	27.71
SSTF	29.50	2.00	84.00	28.62	4.95	2.20	10.40	2.86	19.95	3.20	39.60	13.38
SCAN	26.22	2.00	65.00	22.48	4.62	2.20	8.50	2.25	21.18	3.40	41.60	13.78
C-SCAN	38.20	2.00	199.00	58.84	5.82	2.20	21.90	5.88	21.37	3.20	58.20	19.74
C-LOOK	40.25	2.00	169.00	55.16	6.03	2.20	18.90	5.52	21.02	3.20	48.20	16.92

Figure 2: Summary data for Experiment 1.


Run Data Frame FCFS

Run Data Frame FCFS

From Block	To Block	Request Time	Start Time	End Time	Seek Time	Turnaround Time	Pending Queue
53	98	0.000	0.000	6.500	6.500	6.500	0
98	183	0.000	6.500	17.000	10.500	17.000	1
183	37	0.000	17.000	33.600	16.600	33.600	2
37	122	0.000	33.600	44.100	10.500	44.100	3
122	14	0.000	44.100	56.900	12.800	56.900	4
14	124	0.000	56.900	69.900	13.000	69.900	5
124	65	0.000	69.900	77.800	7.900	77.800	6
65	67	0.000	77.800	80.000	2.200	80.000	7

Count: 8

Figure 3: Run data for the FCFS algorithm in Experiment 1.

In the examples given in most textbooks, a few requests (less than a dozen) are assumed to be pending, and no other requests arrive while these requests are being handled. This, of course, makes it easy to manually trace the algorithms and illustrate how the algorithms differ. To use the simulator with this type of example, create a file containing the list of tracks to be accessed. The simulator also allows you to specify the arrival time of each request. In the textbook examples, the arrival times are all 0.

The simulator is designed to make it easy for students to perform experiments. An *experiment* consists of a number of *runs* in which only a few factors vary. An experiment to compare different disk head scheduling algorithms uses the same requests in each run and only changes the head movement algorithm. Comparing head movement algorithms is only one of a large possible number of experiments. Some are described in Section 8.

The simulator uses three types of algorithms to specify where the sectors are on the disk, the order in which seeks occur, and how long it takes to seek.

Layout
While most textbooks compare the head movement algorithms in terms of cylinder numbers, file systems are typically organized in blocks, where each block corresponds to a given number of sectors. With a uniform layout, the number of sectors per cylinder is fixed. The **Layout** column in Figure 2 indicates that the simulations were done with a **Uniform 1** layout, meaning 1 sector per cylinder. With this layout, sector numbers are the same as cylinder numbers, allowing the simulator to take cylinder numbers as input.

The simulator can also handle a zoned layout. Zoned layouts are common on modern disk drives. In a zoned layout the cylinders are grouped with each group having a different uniform layout. Such a layout allows the disk to be used more efficiently, since more sectors can fit on an outer track than on an inner one.

Seek Movement
Textbooks typically focus on the seek movement algorithm. The simulator supports FCFS, SSTF, SCAN, LOOK, C-SCAN and C-LOOK algorithms, as well as other algorithms including FSCAN. The simulator allows the double buffering technique used by FSCAN to be applied to any of the algorithms. Double buffering is particularly useful with SSTF, as this algorithm can indefinitely postpone a request that is far from the current position.

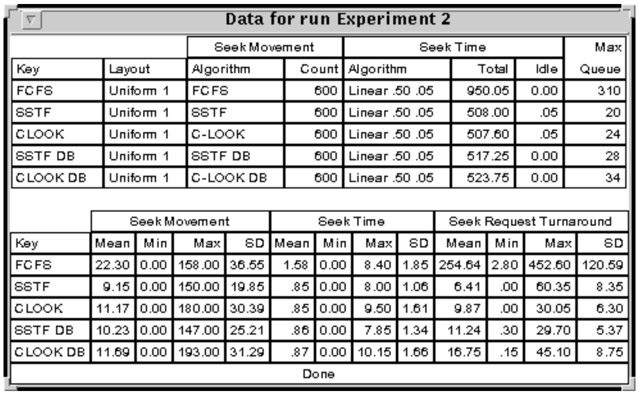
Seek Time
Seek times usually have two components: a fixed time which is independent of the distance the head needs to travel, and a time based on the number of cylinders the head needs to move. The simplest type of algorithm is linear. The algorithm **Linear 2.00 .10** shown in Figure 2 means that a seek takes 2 units of time plus .1 unit per cylinder traveled. The simulator can then calculate statistics based on both distance traveled and time. Figure 2 shows the SSTF algorithm has a mean seek movement of 26.22 cylinders and a mean seek time of 4.40 units of time. The simulator also calculates turnaround time, the time between when a request comes in and when the corresponding seek is finished. Figure 3 shows the movement and time for each individual seek in a run.

4 Beyond the Textbook

A simple list of cylinders that all arrive at time 0 may be sufficient for illustrating how scheduling algorithms work, but a more realistic and much larger set is necessary to compare the performance of different algorithms.

An alternative to specifying a list of sectors is to specify a first arrival time, a distribution of interarrival times, and a distribution of sector references. The simulator allows you to specify several sets of these distributions to simulate different types of concurrent loading.

Experiment 2 compares several algorithms with with two types of references. One set of references accesses a small section of the disk often, as would be the case when a single process is making many random accesses to a single file. The other set of references accesses the disk in a more uniform manner. The five algorithms compared are FCFS, SSTF, C-LOOK, SSTF DB (SSTF with double buffering) and C-LOOK DB. Without the double buffering, SSTF tends to keep the disk head in one area when there are sufficient references to keep it busy. Figure 4 shows a table of summary statistics for Experiment 2.



Data for run Experiment 2

		Seek Movement		Seek Time			Max
Key	Layout	Algorithm	Count	Algorithm	Total	Idle	Queue
FCFS	Uniform 1	FCFS	600	Linear .50 .05	950.05	0.00	310
SSTF	Uniform 1	SSTF	600	Linear .50 .05	508.00	.05	20
CLOOK	Uniform 1	C-LOOK	600	Linear .50 .05	507.60	.05	24
SSTF DB	Uniform 1	SSTF DB	600	Linear .50 .05	517.25	0.00	28
CLOOK DB	Uniform 1	C-LOOK DB	600	Linear .50 .05	523.75	0.00	34

	Seek Movement				Seek Time				Seek Request Turnaround			
Key	Mean	Min	Max	SD	Mean	Min	Max	SD	Mean	Min	Max	SD
FCFS	22.30	0.00	158.00	36.55	1.58	0.00	8.40	1.85	254.64	2.80	452.60	120.59
SSTF	9.15	0.00	150.00	19.85	.85	0.00	8.00	1.06	6.41	.00	60.35	8.35
CLOOK	11.17	0.00	180.00	30.39	.85	0.00	9.50	1.61	9.87	.00	30.05	6.30
SSTF DB	10.23	0.00	147.00	25.21	.86	0.00	7.85	1.34	11.24	.30	29.70	5.37
CLOOK DB	11.69	0.00	193.00	31.29	.87	0.00	10.15	1.66	16.75	.15	45.10	8.75

Figure 4: Experiment 2 has two types of requests.

You can understand the results of this simulation with a simple analytical analysis. The runs were made assuming a seek time of .5 plus .05 times the number of cylinders to move. Requests are generated from two distributions. One chooses cylinders uniformly in the range 40 to 50, with a constant arrival rate of 1 ms for a total of 500 requests. The other distribution chooses cylinders uniformly in the range 0 to 200 every 5 ms for a total of 100 requests. All requests have been made by 500 ms. Exactly 6 new requests come in every 5 ms, or .83 ms per request. As requests come in, they are put in a queue of pending requests. The requests can be handled if the average seek time is less than the average interarrival time.

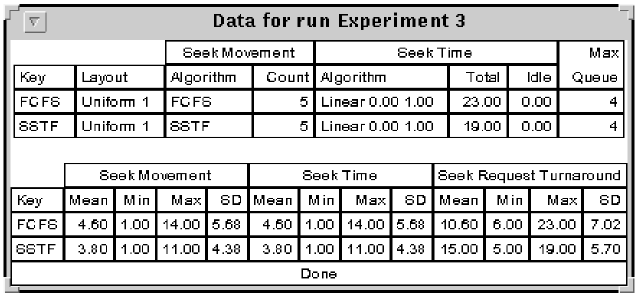
The FCFS algorithm must handle requests in the order they are received. There are two type of seeks. The short seeks stay in the range 40 to 50, average about 3.3 cylinders and have a seek time of about .67 ms. The long seeks are between a cylinder in the range 40 to 50 and a randomly chosen cylinder between 0 and 200. These average about 65 cylinders and have an average seek time of about 3.75 ms. Out of every 6 seeks, 4 are small and 2 are large, giving an average seek time of about 1.7 ms. Since this is greater than the average interarrival time of .83 ms, FCFS cannot handle these requests fast enough. We see from Figure 4 that the mean seek time for FCFS is 1.58, close to our calculated value of 1.7. The maximum size of the queue of pending requests is 310. Other experiments show that the mean seek time remains about the same but the maximum queue size and turnaround time grow indefinitely if requests are allowed to come in at the same rate.

The other algorithms are more stable. As the number of pending requests grows, the average seek time decreases. The number of pending requests increases until the seek time is less than the interarrival time. For example, consider the SCAN algorithm that starts a scan with 18 pending requests. There are about 15 requests in the range 40 to 50 and 3 elsewhere. The 18 requests can be handled in

.5*18 + .05*200 = 19 ms, giving an average seek time of about 1.1 ms. This is slightly greater than the average interarrival time of .83, so 18 pending requests are not enough. How many requests need to be pending to make the average seek time equal to the interarrival time? The answer can be found by solving the equation: .5*Q + .05*200 = Q*.83. This is satisfied by Q = 30. We expect that a slightly smaller queue will do for the LOOK algorithm. As shown in Figure 4, the algorithms other than FCFS have a maximum queue size between 20 and 34 and their mean seek time is almost equal to the average interarrival time. For these algorithms, the numbers do not change much if the simulation is run for longer periods of time.

It is tempting to compare FCFS and SSTF in disk head scheduling to FCFS and SJF in CPU scheduling. In the simplest case, all of the requests are available when the algorithm starts and no new requests come in. In this case it can be shown that SJF scheduling has the smaller (or equal) average waiting time. The same applies to turnaround time which also counts processing time. What can be said about FCFS and SSTF in disk head scheduling? The analog of processing time for disk head scheduling is the amount of time to seek to the requested cylinder. In this case, unlike CPU scheduling, the processing time depends on the current state of the system, that is the current position of the disk head. It is possible for FCFS to have a shorter average turnaround time than SSTF.

Consider Experiment 3, in which the disk head is at cylinder 10 and there are seeks pending to cylinders 5, 16, 17, 18 and 19. FCFS might process these in the order 16, 17, 18, 19, 5 while SSTF would always use the order 5, 16, 17, 18, 19. If the time to seek is equal to the distance, the turnaround times under SSTF are 5, 16, 17, 18, and 19, giving a total of 75. Under FCFS the turnaround times are 6, 7, 8, 9 and 23 for a total of 53. The total head movement for SSTF is shorter (19) than for FCFS (23). However, although SSTF has a smaller total seek time, in this case it has a larger turnaround time than FCFS. Figure 5 shows the simulator results for this simple example.



Data for run Experiment 3

		Seek Movement		Seek Time			Max
Key	Layout	Algorithm	Count	Algorithm	Total	Idle	Queue
FCFS	Uniform 1	FCFS	5	Linear 0.00 1.00	23.00	0.00	4
SSTF	Uniform 1	SSTF	5	Linear 0.00 1.00	19.00	0.00	4

	Seek Movement				Seek Time				Seek Request Turnaround			
Key	Mean	Min	Max	SD	Mean	Min	Max	SD	Mean	Min	Max	SD
FCFS	4.60	1.00	14.00	5.68	4.60	1.00	14.00	5.68	10.60	6.00	23.00	7.02
SSTF	3.80	1.00	11.00	4.38	3.80	1.00	11.00	4.38	15.00	5.00	19.00	5.70

Figure 5: Experiment 3, compares FCFS and SSTF.

5 Logical Sectors

Suppose an operating system has a logical view of a disk that differs from the physical view. The operating system's (logical) view of the disk is described by three numbers, the number of heads, *LogicalHeads*, the number of tracks per surface, *LogicalTracks*, and the number of sectors per track, *LogicalSectors*. The total number of sectors is

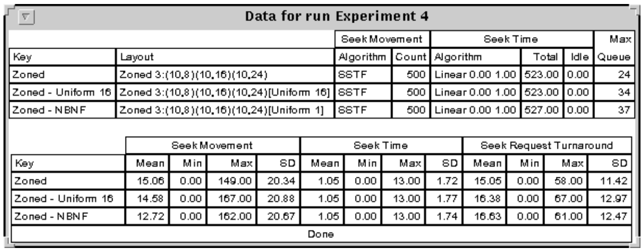
$$\mathit{LogicalHeads} \times \mathit{LogicalTracks} \times \mathit{LogicalSectors}.$$

The physical disk has a number of heads, *PhysicalHeads*, a number of tracks per surface, *PhysicalTracks* and a number of sectors per track, $\mathit{PhysicalSectors}_i$. This last value depends on the track number, i. The total number of sectors on the disk is

$$\mathit{PhysicalHeads} \times \sum_i (\mathit{PhysicalSectors}_i).$$

FCFS, SCAN, and LOOK scheduling are not affected by the difference between the logical and physical views of the disk. However, SSTF scheduling might not behave as well when the operating system does not have a correct physical view of the disk structure.

You can specify separate layout algorithms for the operating system view and the physical view. The operating system view layout is used to decide which block to access next, while the physical system view is used to determine how long it takes to do a seek. Figure 6 shows the results of Experiment 4 in which the operating system thinks the disk layout is uniform, but the actual disk layout has three zones. Three runs are shown. In each run the physical disk has 3 zones of 10 cylinders each. The first zone has 8 sectors per cylinder, the second has 16 and the last has 24. In the first run, the operating system view is the same as the physical view. In the second run, the operating system view is uniform with 10 cylinders of 16 tracks each. The seek request turnaround time is slightly larger when the operating system has an incorrect view. The third run is described in the next section.



Data for run Experiment 4

		Seek Movement		Seek Time			Max
Key	Layout	Algorithm	Count	Algorithm	Total	Idle	Queue
Zoned	Zoned 3:(10,8)(10,16)(10,24)	SSTF	500	Linear 0.00 1.00	523.00	0.00	24
Zoned - Uniform 16	Zoned 3:(10,8)(10,16)(10,24)[Uniform 16]	SSTF	500	Linear 0.00 1.00	523.00	0.00	34
Zoned - NBNF	Zoned 3:(10,8)(10,16)(10,24)[Uniform 1]	SSTF	500	Linear 0.00 1.00	527.00	0.00	37

	Seek Movement				Seek Time				Seek Request Turnaround			
Key	Mean	Min	Max	SD	Mean	Min	Max	SD	Mean	Min	Max	SD
Zoned	15.06	0.00	149.00	20.34	1.05	0.00	13.00	1.72	15.05	0.00	58.00	11.42
Zoned - Uniform 16	14.58	0.00	167.00	20.88	1.05	0.00	13.00	1.77	16.38	0.00	67.00	12.97
Zoned - NBNF	12.72	0.00	162.00	20.67	1.05	0.00	13.00	1.74	16.63	0.00	61.00	12.47

Figure 6: Experiment 4 compares different views of the disk.

6 Logical Block Addressing

In Logical Block Addressing, the operating system communicates with the disk controller using absolute sector numbers. The logical view of the disk is an array of sectors. The operating system does not have any information about the cylinder at the head position. As before, FCFS, SCAN and LOOK scheduling are not affected by this view. SSTF cannot be used since the operating system does not have information on track numbers, but it can approximate this with NBNF (Nearest Block Number First). The penalty in using this instead of SSTF is at most one cylinder per seek, so NBNF performance should be close to that of SSTF. The third run shown in Figure 6 shows the result when the operating system uses NBNF. The seek request turnaround time is slightly larger than when the operating systems has the correct view of the disk structure.

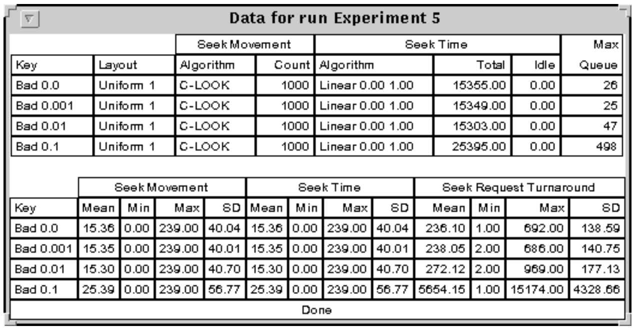
7 Bad Blocks

The operating system can handle bad disk blocks by preventing these blocks from being allocated to a file or by mapping bad blocks to reserved blocks. In either of these cases, the existence of bad blocks does not change how well the operating system understands the disk layout. The only penalty imposed is the added fragmentation of files.

Modern disk drives handle bad sectors transparently. To the operating system, the disk appears to have no bad blocks. Bad blocks are transparently mapped by the disk controller to blocks specifically reserved for this purpose. One method is to keep some spare sectors on each track or cylinder to act as replacements for bad sectors. In this case the mapping does not incur any additional seek penalty but there may be additional rotational latency in accessing mapped sectors.

Another possibility is for the reserved sectors to be on cylinders reserved for replacements of all bad sectors. If this is transparent to the operating system, the operating system may make an incorrect decision when picking the next block to access. Of the algorithms discussed here, only FCFS is not susceptible to this type of error.

The simulator allows you to specify a fraction of bad blocks to be mapped and a distribution for the location of the mapped blocks. The simulator assumes that the operating system does not know that a mapping has taken place. The simulator uses the original location of the block to determine which block to access next, but it uses the actual location of the block to determine seek times. Figure 7 shows the results of Experiment 5, which uses the C-LOOK algorithm with different fractions of bad blocks. When the fraction of bad blocks is one percent, the average seek time does not change much and the seek turnaround time increases by about 15 percent. For 10 percent bad blocks, the average seek time increases by about 67 percent and the seek request turnaround time increases by a factor of 20.


Data for run Experiment 5

		Seek Movement		Seek Time			Max
Key	Layout	Algorithm	Count	Algorithm	Total	Idle	Queue
Bad 0.0	Uniform 1	C-LOOK	1000	Linear 0.00 1.00	15355.00	0.00	26
Bad 0.001	Uniform 1	C-LOOK	1000	Linear 0.00 1.00	15349.00	0.00	25
Bad 0.01	Uniform 1	C-LOOK	1000	Linear 0.00 1.00	15303.00	0.00	47
Bad 0.1	Uniform 1	C-LOOK	1000	Linear 0.00 1.00	25395.00	0.00	498

	Seek Movement				Seek Time				Seek Request Turnaround			
Key	Mean	Min	Max	SD	Mean	Min	Max	SD	Mean	Min	Max	SD
Bad 0.0	15.36	0.00	239.00	40.04	15.36	0.00	239.00	40.04	236.10	1.00	692.00	138.59
Bad 0.001	15.35	0.00	239.00	40.01	15.35	0.00	239.00	40.01	238.05	2.00	686.00	140.75
Bad 0.01	15.30	0.00	239.00	40.70	15.30	0.00	239.00	40.70	272.12	2.00	969.00	177.13
Bad 0.1	25.39	0.00	239.00	56.77	25.39	0.00	239.00	56.77	5654.15	1.00	15174.00	4328.66

Figure 7: Experiment 5 shows the effect of bad blocks on the C-LOOK algorithm.

8 Curriculum

The simulator can be used for in-class demonstrations of the traditional scheduling algorithms. Students can also use it to perform experiments outside of class. The tutorial that is available on the web site can be used as a standalone explanation without requiring any classroom discussion. If class time is tight, students can go through the tutorial after reading a discussion of disk head scheduling in a standard textbook and then be assigned problems using the simulator similar to the ones below.

- Compare standard scheduling algorithms such as FCFS, SSTF, LOOK and C-LOOK for a disk with a simple layout and fixed linear seek time. Generate requests uniformly throughout the entire disk.
- How does the pattern of requests affect the performance of a scheduling algorithm? Choose an algorithm and vary the request pattern. The simplest pattern is uniformly distributed. Compare this with a pattern in which requests are of two types, each type concentrated in a particular area of the disk.
- How are each of the standard scheduling algorithms affected by disk load? Which algorithms degrade gracefully as the load is increased? Look at the maximum queue size and average seek and turnaround times. Does the average seek time increase or decrease under heavy load?
- Suppose the operating system assumes a fixed number of sectors per track but the disk has a zoned layout. Which of the standard scheduling algorithms are most affected by the difference between the logical and physical layout of the disk?
- The seek time consists of two parts, one that is independent of the distance to seek and one that depends on this distance. Which algorithms are most affected by each of these parts?
- How do bad blocks affect performance? Is there a critical percentage of bad blocks beyond which the performance degrades rapidly? Does this percentage depend on the seek movement algorithm?

9 Discussion

The simulator is written in Java and can be run remotely from a browser or the code can be downloaded and run locally [3]. The web site allows users to run the five experiments described in this paper from a browser without installing any software. A step-by-step tutorial is also available that allows students to learn how to use the simulator without additional help. A user's guide gives complete documentation for those who want to explore disk head scheduling in more detail.

The disk head simulator has a number of limitations. It is difficult to simulate real workloads in which most accesses are to files that may be stored contiguously. The simulator ignores both rotational latency and transfer time, and it is limited to runs of a few thousand block accesses. However, the simulator does allow students to easily design experiments to test a variety of disk head scheduling algorithms and to gain insight into their operation.

10 Acknowledgments

This work has been supported by an NSF grant: *A Course in Experimental Techniques for Computer Science Majors: Proof of Concept*, DUE-0088769.

References

[1] Denning, P. J., “Effects of scheduling on file memory operations,” *AFIPS Spring Joint Computer Conference*, April 1967, pp. 9–21.

[2] Nutt, G., *Operating Systems, A Modern Perspective, Second Edition,* Addison-Wesley, 2000.

[3] Robbins, S., A disk head scheduling simulator, 2003. Online. Internet. Available WWW: **http://vip.cs.utsa.edu/nsf/disk/**

[4] Silberschatz, A., Galvin, P. B. and Gagne, G., *Operating System Concepts, Sixth Edition,* John Wiley and Sons, Inc, 2002.

[5] Stallings, W., *Operating Systems, Second Edition,* Prentice Hall, 1995.

[6] Tanenbaum, A., *Modern Operating Systems, Second Edition,* Prentice Hall, 2001.

[7] Thomasian, A. and Chang, L., “Disk scheduling policies with lookahead,” *ACM Sigmetrics Performance Review*, vol 30, 2002, pp. 31–40.