5-2000

A Study of Disk Performance Optimization.

Richard Scott Gray
East Tennessee State University

A STUDY OF DISK PERFORMANCE OPTIMIZATION

_____

A Thesis

Presented to

the Faculty of the Department of Computer Science

East Tennessee State University

_____

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science in Computer Science

_____

by

Richard S. Gray

May 2000

APPROVAL

This is to certify that the Graduate Committee of

Richard S. Gray

met on the

3rd day of March, 2000.

The committee read and examined his thesis, supervised her defense of it in an oral examination, and decided to recommend that her study be submitted to the Graduate Council, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

_____
Chair, Graduate Committee

_____

_____

_____

Signed on behalf of
the Graduate Council

_____
Dean, School of Graduate Studies

A B S T R A C T

A STUDY OF DISK PERFORMANCE OPTIMIZATION

by

Richard S. Gray

Response time is one of the most important performance measures associated with a typical multi-user system. Response time, in turn, is bounded by the performance of the input/output (I/O) subsystem. Other than the end user and some external peripherals, the slowest component of the I/O subsystem is the disk drive.

One standard strategy for improving I/O subsystem performance uses high-performance hardware like Small Computer Systems Interface (SCSI) drives to improve overall response time. SCSI hardware, unfortunately, is often too expensive to use in low-end multi-user systems. The low-end multi-user systems commonly use inexpensive Integrated Drive Electronics (IDE) disk drives to keep overall costs low. On such IDE based multi-user systems, reducing the Central Processing Unit (CPU) overhead associated with disk I/O is critical to system responsiveness.

This thesis explores the impact of PCI bus mastering Direct Memory Access (DMA) on the performance of systems with IDE drives. DMA is a data transfer protocol that allows data to be sent directly from an attached device to a computer system's main memory, thereby reducing CPU overhead. PCI bus mastering allows modern IDE disk controllers to manipulate main memory without utilizing motherboard-resident DMA controllers.

Using a series of experiments, this thesis examines the impact of PCI bus mastering DMA on IDE performance for synchronous I/O, relative to Programmed Input/Output (PIO) and SCSI performance. Experiment results show that PCI bus mastering DMA, when used properly, improves the responsiveness and throughput of IDE drives by as much as a factor of seven. The magnitude of this improvement shows the importance of operating system support for DMA in low-end multi-user systems. Additionally, experimental results demonstrate that performance gains associated with SCSI are dependent on system usage and operating system support for advanced SCSI capabilities. Therefore, under many circumstances, high-performance SCSI drives are not cost effective when compared with IDE bus mastering DMA capable drives.

CONTENTS

Page

Chapter Page

Chapter Page

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

C H A P T E R 1

INTRODUCTION

Response time, the time required to service a single user request, is one of the most important performance measures associated with a typical multi-user system. Response time, in turn, is bounded by the performance of the input/output (I/O) subsystem. Other than the end user and some external peripherals, the slowest component of the I/O subsystem is the disk drive. One standard strategy for improving the performance of disk drives of all kinds is to use Direct Memory Access (DMA) to reduce Central Processing Unit (CPU) overhead. This thesis reviews several approaches to improving disk performance and examines the use of Direct Memory Access (DMA) to reduce Central Processing Unit (CPU) overhead in detail.

Direct Memory Access (DMA)

The common approach for improving I/O subsystem performance uses high-performance hardware like Small Computer Systems Interface (SCSI) drives and controllers. SCSI subsystems, unfortunately, are typically too expensive to incorporate into low-end multi-user systems. Such low-end systems often use IDE disk drives to reduce overall system cost.

DMA is a data transfer protocol that allows external computer devices to transfer data to and from main memory without CPU intervention. Modern system motherboards and IDE disk controllers use a technique for implementing DMA known as Peripheral

Component Interconnect (PCI) bus mastering to move data to and from main memory without the use of motherboard-resident DMA controllers.

Using a series of experiments, this thesis examines the impact of PCI bus mastering DMA on IDE performance for synchronous I/O, relative to Programmed Input/Output (PIO) and SCSI performance. Experiment results reveal that PCI bus mastering DMA, when used properly, improves the responsiveness and throughput of IDE drives by as much as a factor of seven. Additionally, experimental results demonstrate that performance gains associated with SCSI are dependent on system usage and operating system support for advanced SCSI capabilities. Therefore, under many circumstances, high-performance SCSI drives are not cost effective when compared with IDE bus mastering DMA capable drives.

Overview

Chapter 2, Hard Drive Technology, discusses current hard drive technology including mechanical drive components, drive transfer protocols, drive performance specifications, and disk-scheduling algorithms. Chapter 3, Linux File System Support, provides an overview of Linux file system support necessary to understand file system factors influencing drive performance. Experiments conducted for this thesis were executed on the Linux operating system. Chapter 4, Smart Disks, describes various disk drive data placement optimization strategies primarily focusing on an adaptive technique known as Adaptive Block Rearrangement (Akyurek & Salem, 1995).

Chapter 5, Performance Test Results, discusses experiments conducted as part of this thesis to evaluate benefits associated with DMA. Initially, computer hardware

specifications, operating system specifications, and the testing procedure are described to ensure that all experiments are reproducible. A detailed description of each performance test (benchmark) utility is given prior to any analysis of results obtained using the benchmark. Chapter 6, Summary, reiterates lessons learned in the development of this thesis and recommends potential areas for future research.

CHAPTER 2

HARD DRIVE TECHNOLOGY

Recent improvements in hard disk technology have increased the storage capacities for hard drives, with storage capacities increasing at a rate of 60 to 80 percent compounded annually (Ruemmler & Wilkes, 1994). However, disk performance improvements have lagged, increasing at a rate of only 7 to 10 percent compounded annually (Ruemmler & Wilkes, 1994). In contrast, microprocessor speeds have been increasing dramatically with typical speedups of 40 to 60 percent compounded annually (Ruemmler & Wilkes, 1994). In order to improve the speed of future computer systems, the above disparity between the performance of the CPU and the hard drive must continue to be addressed.

This chapter begins by presenting an overview of the various components of the hard drive. Additionally, recent innovations in hard drive technology and the impact of these advances on hard drive performance are addressed. Next, a discussion of the various hard drive performance specifications is presented. Disk scheduling attempts to optimize the order in which the read-write requests sent to a disk drive are satisfied. An understanding of the various disk-scheduling algorithms provides insight into the inner workings of the hard drive. This section concludes with a discussion of disk head-scheduling algorithms.

Structural Disk Components

A hard drive consists of one to twenty aluminum-alloy or glass platters mounted on a rotating spindle (see Figure 17, Appendix C). Both surfaces of a platter are typically covered with a magnetic substance. Associated with each surface is a read-write head used

to retrieve data from and record data to the platter's surface. A magnetic field generated by the read-write head magnetizes regions on the disk's surface, thereby recording information on the disk. The read-write heads are constructed of a U-shaped piece of conductive material wrapped with coils of wire. When an electrical current is sent through the coils, a magnetic field is generated in the gap of the read-write head. The polarity of this magnetic field is dependent on the electrical current's direction of flow (Ruemmler & Wilkes, 1994). Each read-write head is attached to a lever (arm) that is pivoted on a rotational bearing. These disk arms are attached to a single rotation pivot. As the read-write head floats on a cushion of air approximately 3 microinches (.0762000000001 micrometers) above its corresponding platter surface, the magnetic field generated by the read-write head is used to produce a magnetic pattern on the disk's surface. An electrical current is generated, if a conductor is passed through a changing magnetic field. The term flux refers to a magnetic field that has a specific direction. By detecting changes in the polarity of the alignment of magnetic particles on a platter's surface, referred to as flux transitions, the read-write heads convert the magnetic patterns recorded on the media to an electrical current. This current is amplified and processed to reconstruct the stored data pattern by a single channel (found on the controller) that is shared among all of the disk heads.

Each platter surface is divided into tracks (concentric circles) on which data is stored. The set of tracks directly above and below a track is known as a cylinder. Platters rotate in lockstep on a spindle motor at speeds ranging from 3,600 to 10,000 revolutions per minute (RPM). It is typical for a platter to contain more than 14,500 tracks per inch (TPI). Most of the recent disk capacity gains have been brought about by packing adjacent tracks closer

together (Ruemmler & Wilkes, 1994). Tracks are subdivided into sectors, each of which is usually large enough to contain 512 bytes of data in addition to required addressing information. A sector represents the smallest addressable location on a disk drive. This organization is intended to facilitate storage and retrieval of data.

Each read-write head is held in place by a single actuator arm. This arm serves to move the read-write heads to the correct track on which requested data resides. The time associated with moving the read-write head is referred to as seek time. Once a read-write head arrives over the correct track, it typically waits for the sector containing the sought after data to rotate underneath it. This delay is referred to as the rotational latency.

Associated with each disk is a controller. The controller consists of a specialized microprocessor, some buffer memory, and an appropriate bus interface. The buffer memory is composed of dual-ported static random access memory (RAM) and is used for both disk-to-host (read) and host-to-disk (write) data transfers. When the CPU issues a data request, the disk drive retrieves the requested sectors and reads them into the controller's cache memory. On some drives, the read-write head does not begin reading until the requested sector rotates directly beneath it. Other disk drives take a more optimistic approach. In such cases, the read-write head immediately begins reading once it is positioned over the correct track. In both cases, the read-write head continues reading until the cache buffer is full (Ruemmler & Wilkes, 1994).

On older hard drives, there are a fixed number of sectors per track. This results in a loss of potential storage space because the outer tracks are longer. Modern disk drives use a formatting technique called Multiple Zone Recording (MZR) (Quantum Corporation,

2000). MZR formatting groups a disk's adjacent cylinders into 3 to 20 disjoint sets (zones) such that all tracks in a zone have the same data storage capacity. The tracks in a zone on the outer parameter of the disk contain more data than the tracks in a zone on the inner region. MZR impacts the hard drive in two significant ways. Dividing the outer tracks into more sectors allows data to be distributed more evenly across each platter. More significant is the impact of MZR on the disk-to-buffer ratio. Data can be read faster from the outer zones because the tracks contained in these zones hold more data (Quantum Corporation, 2000). Information is first recorded on the outer most cylinders. Once all of the outer tracks are filled with data, the read-write heads move inward and begin storing data.

IDE Transfer Protocols

Two standard methods for transferring data between a computer system's external devices and main memory are programmed input/output (PIO) and direct memory access (DMA). The important similarity between PIO and DMA involves their use of interrupts, rather than CPU-based polling, to detect and respond to I/O events. The important difference between PIO and DMA lies in the procedure for moving data between devices and primary memory. PIO controllers rely on the CPU to move data in and out of main memory. A PIO transfer of N words requires N loads, N stores, and N interrupts to complete (Kozierok, 2000). DMA devices use specialized controllers to transfer blocks of words without CPU intervention. PCI bus mastering, for example, allows IDE disk controllers to manipulate main memory directly without utilizing motherboard-resident DMA controllers. Initially, the CPU sets DMA registers that specify the starting address

to/from which data is to be stored/retrieved and the number of bytes to be stored/retrieved. Upon completion of the data transfer, the DMA controller generates an interrupt.

One standard characterization of disk subsystem performance, external transfer (burst) rate, measures the speed at which data can be exchanged between system memory and the cache on the hard drive's interface. The various PIO and DMA modes along with their corresponding external data transfer rates are listed in Table 1.

The internal data transfer rate or sustained transfer rate refers to the rate that data can be physically read from the hard drive. For sustained reads of any reasonable size, the overall data transfer rate will be the same as the sustained transfer rate. Since modern disk drives utilize MZR, outer disk tracks, which have more sectors than inner tracks, offer a high transfer rate. For this reason, disk drive manufacturers commonly advertise a maximum sustained transfer rate obtained during burst mode transfers from the outer part of the disk (Kozierok, 2000).

Table 1

IDE Maximum Data Transfer Rates

Protocol	Maximum Transfer Rate (MB/Second)
PIO	
Mode 0	3.3
Mode 1	5.2
Mode 2	8.3
Mode 4	11.4
DMA	
Single Word 0	2.1
Single Word 1	4.2
Single Word 2	8.3
Multiword 0	4.2
Multiword 1	13.3
Multiword 2	16.6
Multiword 3	33.3

Note. IDE = Integrated Drive Electronics, MB = megabytes. Source adapted from Kozierok, C. M. (2000). The PC Guide. Site Version 1.11.0. Unpublished manuscript. Retrieved February 7, 2000 from the World Wide Web: http://www.pcguide.com/topic.html.

The SCSI Standard

The Small Computer Systems Interface (SCSI) standard is a system level interface for connecting external devices to a computer system. Each SCSI host adapter provides a high speed SCSI bus to which up to seven SCSI devices may be attached. Each peripheral on a

SCSI bus, including the host adapter, is assigned a SCSI ID ranging from zero to seven. The three SCSI standards defined by the American National Standards Institute (ANSI) are SCSI-1 (standard), SCSI-2 (Fast SCSI), and SCSI-3 (Ultra SCSI). Table 2 describes the maximum data transfer rates associated with each SCSI standard. As depicted in Table 2, the use of a 16-bit (Wide SCSI) bus doubles the maximum data transfer rate associated with each SCSI protocol.

Table 2.

SCSI Maximum Data Transfer Rates

Bus Width	Standard	Fast SCSI	Ultra SCSI	Cable
8-bit	5 MB/sec	10 MB/sec	20 MB/sec	50-pin
16-bit	10 MB/sec	20 MB/sec	40 MB/sec	68-pin

Note. SCSI = Small Computer Systems Interface, MB = megabytes, sec = second. Source adapted from Risley, D. (n. d.). SCSI. Darien, CT: Internet.com Corp. Unpublished manuscript. Retrieved February 14, 1999 from the World Wide Web: http://www.hardwarecentral.com/hardwarecentral/tutorials/36/3/.

Hard Disk Performance Specifications

Seek Time

As defined earlier, seek time refers to the time required to move a read-write head from one track to another. However, a seeking operation is more complex than the

previous definition implies. A typical seek is composed of four phases. While in the speedup phase, the disk arm accelerates until it reaches half the total seeking distance or a predefined maximum speed. The disk arm then enters the coast phase in which it moves at a predefined maximum speed. Next, the disk arm enters the slowdown phase in which it slows until the desired disk track is approached. Finally, the disk controller adjusts the read-write head to access the desired track during the settle phase (Ruemmler &Wilkes, 1994).

Short seeks spend most of their time in the speedup phase and may never enter the coasting phase. Very short seeks ranging from two to four cylinders are dominated by settle time. In such cases, the read-write head may just settle into the correct track. As hard disk densities have increased, the diameter of platters on which they store data have decreased. A smaller platter size serves to lessen the worst and average seek times. Additionally, a smaller platter size increases the percentage of total seek-time attributed to the settling phase (Ruemmler & Wilkes, 1994).

On a modern disk, only two milliseconds (ms) are required to move the read-write head between adjacent cylinders. A seek from the outermost cylinder to the innermost cylinder is referred to as a full-stroke. On a modern disk drive, a full-stroke seek typically requires 20 ms. Manufacturers often quote the average seek time as a major performance specification. However, there is no clear definition as to what constitutes an average seek. Some manufacturers quote the one-third-seek time as the average seek time. Others quote the full-stroke time divided by three. The problem with both of these measures is that they assume that random seeks are common. Actually, random seeks are rare since disk

accesses tend to exhibit locality over time and space, short seeks are far more common. Perhaps the best method for approximating the average seek time is to calculate the required seek time to each track and then divide the value by the total number of tracks (Ruemmler & Wilkes, 1994).

Rotational Latency

Once the read-write head is positioned onto the correct cylinder, it must wait for the requested sector to rotate beneath it. This waiting time is referred to as rotational latency. In the worst case, the head must wait for the platter to complete one full rotation. This occurs if, upon the read-write head's arrival onto the correct track, the sought after sector has just rotated past the read-write heads location. On average, the read-write head must wait one half of a disk rotation for the correct sector to rotate beneath it. This average rotational latency is the performance specification quoted by disk manufacturers (Ruemmler & Wilkes, 1994).

In the past, most disk-based optimizations research was concerned with the reduction of seek times. However, the gap between seek time and rotational latency has decreased. Table 3 compares average the rotational latency and average seek time of a modern drive to corresponding values of its counterpart seven years past.

As indicated above in Table 3, the absolute difference between average seek time and average rotational latency has decreased over the last seven years. The percentage of the combined average rotational latency plus average seek time attributed to rotational latency is 32 percent on the Fujitsu drive and 37 percent on the Medalist Pro. This is because the average seek time has decreased at a greater rate than has the rotational latency over the last

seven years. The simplest way to decrease rotational latency is to increase the disk's rotational speed is depicted in Table 4.

Table 3

Hard Drive Comparison

Drive Characteristics	Medalist Pro (1997)	Fijitsu Eagle (1990)
Cylinders	6,536	840
Tracks per cylinder	10	20
Sectors per track	63	67
Bytes per sector	512	512
Average seek time (ms)	9.5	18
Average Rotational Latency (ms)	5.58	8.3

Note. ms = millisecond. Sources adapted from Seltzer, M., Chen, P., and Ousterhout, J. (1990). Disk Scheduling Revisited. Proceedings 1990 Winter Usenix Conference, Washington, D.C., 313-324. Seagate Technology, Inc. (1997). Medalist Pro Specifications. (Publication No. 32652-001, Rev. A). Scotts Valley, CA: Seagate Technology, Inc. Retrieved September 15, 1997 from the World Wide Web: http:/www.seagate.com:80/support/disc/manuals/ata/6450pma.pdf.

Table 4

Impact of Rotational Speed on Rotational Latency

Rotational Speed (RPM)	Rotational Latency (ms)
3,600	8.3
4,500	6.7
5,400	5.7
6,300	4.8
7,200	4.2

Note. RPM = revolutions per minute; ms = millisecond. Sources adapted from Seltzer, M., Chen, P., and Ousterhout, J. (1990). Disk Scheduling Revisited. Proceedings 1990 Winter Usenix Conference, Washington, D.C., 313-324. Seagate Technology, Inc. (1997). Medalist Pro Specifications. (Publication No. 32652-001, Rev. A). Scotts Valley, CA: Seagate Technology, Inc. Retrieved September 15, 1997 from the World Wide Web: http:/www.seagate.com:80/support/disc/manuals/ata/6450pma.pdf.

Increasing a disk's rotational speed causes it to run hotter, produces more wear on the drive's movable components, and increases power consumption (Ruemmler & Wilkes, 1994). Cylinder skewing is another technique used to decrease rotational latency. The idea behind cylinder skewing is to stagger the beginning of each track to allow for the worst-case track switch time. Head skewing has a similar purpose to that of cylinder skewing. The locations of the sectors across the platters are adjusted to offset for the worst-case head switch.

Transfer Rates

Once a read-write head is correctly positioned over the required sector, the data must be processed and transferred to the CPU. The transfer process can be subdivided into two distinct operations: the time required for the hard drive to pass data to its controller and the time required for the disk controller to pass the data to the CPU. Data transfer rates are described in terms of megabytes (MB).

A disk controller's buffer memory serves to increase the host transfer rate by hiding the asynchrony between the bus and the disk drive (Ruemmler & Wilkes, 1994). Consider how a disk drive would conduct a read operation without any buffer memory. If the disk drive is ready to transfer but the host interface is not free, without any controller buffer memory, one full disk rotation would be required before the next transfer could be made. Write requests can also be buffered. The degree to which write buffering can overlap read buffering is specific to the disk drive controller.

Caching

The functionality of the controller's buffer memory has been extended to include caching for both reads and writes. Lower memory prices have led to a significant increase in the importance of cache buffering. Usually, manufacturers provide little, if any, information as to how a controller's buffer memory is managed. Some basic cache buffering algorithms are discussed below.

Read-ahead Buffering

The term read-ahead buffering describes the process of actively retrieving and caching data in anticipation of future requests. If a requested data item is found in the cache then the request can be satisfied immediately. The only overhead is the time required for the controller to detect the hit and transmit the data to the CPU. This algorithm relies heavily on the Principle of Locality.

On-arrival Read-ahead

On-arrival read-ahead caching is designed to minimize rotational latency. Once the read-write head arrives on the correct track, the entire track is immediately read into the buffer cache. The benefits received from this algorithm are dependent on two factors: the track length and the requested file's size. As the track length increases and the file size decreases, benefits associated with this algorithm decrease (Ruemmler & Wilkes, 1994). Over the last few years, the computer industry has experienced a multimedia explosion. Multimedia files tend to be extremely large. The recent multimedia explosion, the advent of MZR, and the increased importance of rotational latency all combine to make read-ahead caching an appealing algorithm.

Read-ahead in Zero

Currently, the most used buffering algorithm is known as read-ahead in zero (Ruemmler & Wilkes, 1994). This approach is similar to on-arrival read-ahead caching. The distinction is that the entire track is not necessarily read into the cache. Instead, the drive continues to read from the point where the host read request left off. This algorithm

is optimal when accessing sequential data (Ruemmler & Wilkes, 1994). The read-ahead in zero question strategy should not cross track and/or cylinder boundaries if a system generates a large number of random seeks. Once a disk drive has initiated a read-write head switch or a track switch, the operation cannot be aborted. If an unrelated request arrives during the time in which one of the above operations is taking place, the request must wait for the operation to complete.

Read-ahead Cache Segmentation

Typically, a read-ahead cache is segmented to allow for more than one sequential read stream (Ruemmler & Wilkes, 1994). For instance, a 512-kilobyte (KB) cache may be broken up into 16 separate 32 KB segments. If the buffer cache is not segmented in such a manner, then these multiple read streams become interleaved in the buffer, severely degrading cache performance.

Write Caching

It is important to remember that a cache buffer is constructed of volatile memory. To ensure that data is not lost the write cache must be checked for read hits. This is because the write cache contents may not have been updated to disk. Therefore, the data in the write cache is always treated as the primary copy.

Disk Scheduling

In disk systems that use write caching, the controller must check the write cache for read hits before accessing disk. Disk controllers that use write-caches utilize write-back

rather than write-through caching to enhance performance, accordingly, the most recent copy of any data block may reside in the cache, rather than on disk.

Disk scheduling algorithms attempt to order system read-write requests in such a manner as to produce a low mean waiting time, high disk utilization (average disk I/O time), and a low variance between response times. Such algorithms must also guarantee fair response times and thereby prevent starvation of any single request. Most algorithms fail to achieve all these goals, since throughput and response time conflict. Classic scheduling algorithms attempt to achieve performance goals by minimizing seek times. Rotational latency, however, now accounts for a greater percentage of the total I/O time. Research conducted by Seltzer, Chen, and Ousterhout (1990) introduced several disk-scheduling algorithms focusing on the minimization of both seek time and rotational latency.

The Simulator

The simulator that Seltzer et al. (1990) used to measure disk scheduler performance was modeled after the Fujitsu M2361A Eagle disk drive. Specifications for the M2361A drive are listed in Table 3.

Seltzer et al. (1990) ignored the CPU time required to process each request under the assumption that this processing can be overlapped with the I/O operation. The authors focused, rather, on the impact of queue size on scheduling. Previous studies had focused primarily on small task queues, with no more that 50 tasks on average. Seltzer, Chen, and Ousterhout (1990, January) used task queues of varying lengths ranging from one to 100.

First Come First Serve (FCFS)

FCFS is the simplest of all head-scheduling algorithms. The idea is to service disk read-write requests in the same order as they are generated. This algorithm is fair and offers a low variance between response times. FCFS ignores the Principle of Locality. Seltzer et al. (1990) demonstrated that disk utilization under this algorithm was independent of the task queue size, with average I/O time equal to predicted average seek time (18 ms) plus predicted average rotational latency (8.3 ms). Any scheduling algorithm with a queue length of one reverts to FCFS. The FCFS model was used as a base line for addressing other algorithms.

Shortest Seek First (SSF)

SSF selects the next request to service based on required seek distance ignoring rotational latency. This algorithm provides high disk utilization and a low average response time. One problem with SSF is that it tends to discriminate against requests located on the outer and inner cylinders of the disk. Additionally, variance between response times can be large and there is a potential that one or more tasks will experience starvation. The test results indicated that as the queue length grows, disk utilization increases.

SCAN and CSCAN

Both SCAN and CSCAN are variants of SSF. SCAN works much like SSF except it looks for the task with the shortest seek distance in a specific direction. The SCAN algorithm favors tracks in the middle of the disk. Seltzer et al. (1990) found the

performance of SCAN is very similar to that of SSF. CSCAN was designed to address the discrimination problem associated with the SCAN algorithm. Once a scan has completed, a seek is made back to the first pending request in the starting direction. The CSCAN algorithm promotes fairness and provides a low response time variance.

Shortest Time First (STF)

STF optimizes seek time and rotational latency, selecting the read-write request with the shortest I/O time. This algorithm selects the read-write request with the shortest I/O time. Seltzer et al. (1990) demonstrated that STF provides utilization almost identical to that of CSCAN. However, the maximum response time increases dramatically as the queue length increases. STF is the most costly algorithm reviewed thus far in terms of CPU utilization. This is due to the complexity of the algorithm, which is a function of both cylinder position and rotational position.

Grouped Shortest Time First (GSTF)

GSTF combines STF with several strategies used in other scan algorithms. GSTF first partitions a disk into groups of adjacent cylinders. The STF algorithm is applied in each cylinder group, servicing all tasks in that group before moving to the next cylinder group.

The two variables that determine the performance of GSTF are the number of cylinders contained in a cylinder group, and the length of the task queue. Seltzer et al. (1990) found that increasing the number of cylinders per cylinder group increases utilization and maximum response time. There is a potential for starvation if a particular

cylinder group becomes saturated with requests. A variation of GSTF freezes the request queue for a cylinder group once the read-write heads have begun address requests in that cylinder group. This strategy is designed to prevent starvation.

Weighted Shortest Time First (WSTF)

This algorithm represents another variation of the STF algorithm. WSTF applies an aging function to the computed I/O times. As a request waits on the task queue, the aging function is adjusted such that the task becomes more likely to be selected. The weighted time function used by Seltzer et al. (1990) is defined in Figure 1.

Let Tw be the weighted time
Treal be the actual I/O time
M be the maximum response time
E be the elapsed time since this request arrived

$$Tw = Treal \times (M - E) / M$$

Figure 1. Aging function applied to input/output times by Weighted Shortest Time First algorithm. Source adapted from Seltzer, M., Chen, P., and Ousterhout, J. (1990). Disk Scheduling Revisited. Proceedings 1990 Winter Usenix Conference, Washington, D.C., 313-324.

As the task waits on the queue, the WSTF weight factor depicted in Figure 1 becomes smaller, and the request is more likely to be serviced. WSTF produced remarkable test results achieving an average I/O time within 1 to 2 percent of STF's I/O time.

Conclusions

Substantial disk performance increases can be gained with disk scheduling algorithms. The fact that an algorithm works well under one workload does not mean that it will do so under differing workloads. An increase in throughput does not necessarily result in an equivalent decrease in responsiveness. The real problem arises when one tries to achieve a hundred percent of both throughput and responsiveness. This is impossible.

C H A P T E R 3

LINUX FILE SYSTEM SUPPORT

Initially, the Minix file system was used as the Linux file system. The Minix file system, however, had two major limitations: a maximal file system size of 64 megabytes and a fourteen-character limit on the file name size. These limitations led to the development of new file systems for Linux (Card, Ts'o, & Tweedie, 1994). Chris Provenzano developed the Virtual File System (VFS), depicted in Figure 2, as a basis for furthering file systems development. VFS serves as an indirection layer designed to handle file oriented system calls. File systems supported by Linux communicate with the VFS through a common software interface (Rusling, 1999). Structure dependent operations are then redirected to the appropriate file system code by the VFS. Linus Torvalds rewrote the VFS layer before integrating it into the Linux kernel.

The Extended File System (Extfs), introduced by Remy Card in April 1992, was the first new Linux file system to utilize the VFS layer. Extfs increased the maximal file system size to 2 gigabytes and the maximal file name size to 255 characters. Extfs, nevertheless, has its own limitations. This new file system did not support separate access, inode modification, or data modification time stamps. Another problem with Extfs was its use of linked lists to track free blocks and inodes. As a file system aged, linked lists became unsorted. As noted by Card et al. (1994), these disordered free lists, in turn, caused Extfs to become fragmented.

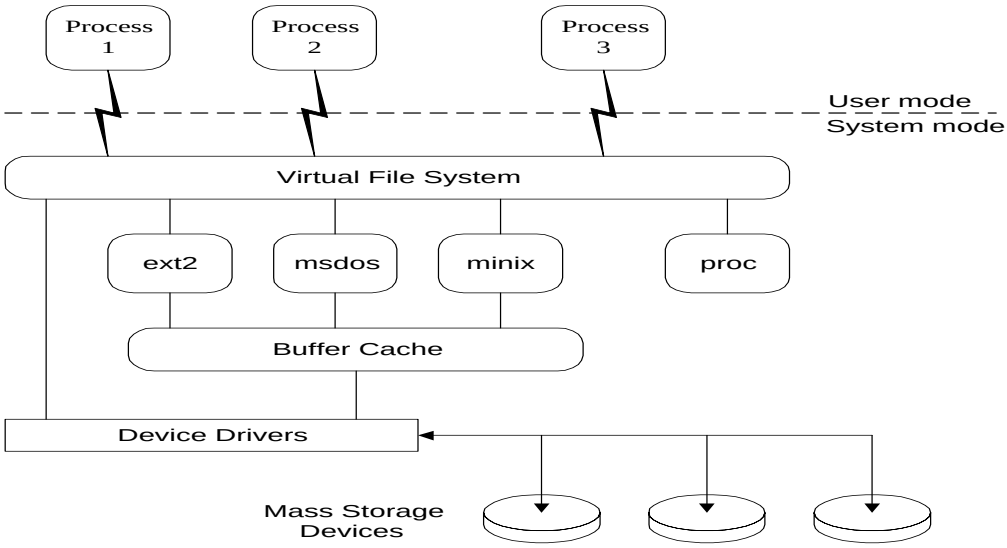


Figure 2. The Linux Virtual File System Layer. Source adapted from Beck, M., Bohme, H., Dziadzks, M., Kunitz, U., Magnus, R., & Verworner, D. (1996). Linux Kernel Internals, Reading, MA: Addison Wesley Publishing Company, Inc.

Two new file systems, Xiafs and the Second Extended File System (Ext2fs) were introduced in January 1993, to remedy Extfs's problems. Xiafs, designed by Stephen Tweedie, was based heavily on the Minix file system. Although Xiafs addressed Extfs's deficiencies, the Xiafs design made future enhancements difficult. Ext2fs, on the other hand, was designed by Remy Card to evolve (Cart et al., 1994). Hooks were placed in the Ext2fs code to reserve space for future enhancements. As bugs were located and removed from Ext2fs it became, and remains, the defacto Linux file system (Cart et al.). Today, Xiafs is rarely used.

Basic Unix File System Concepts

Inodes

Unix implements files as tree-structured objects headed by a control block known as an inode. Each file is represented by an inode describing the file's type, ownership, timestamps, and data block pointers. Possible file types include ordinary, directory, special device, etc. Group and individual ownership associated with the file are listed in the inode. Timestamps specify when the file was created, last accessed, and last modified.

Figure 3 shows a simplified version of an inode as found in Ext2fs. A file's data is found in the leaf blocks of its tree, while the non-leaf blocks contain various pointers. A direct pointer simply points to a file data block. There are twelve direct pointers, one indirect pointer, one double indirect pointer, and one triple indirect pointer in each inode. The block size ranges from 1024 bytes to 8192 bytes. At file system creation, the user determines the block size. If no block size is specified the default block size is typically 1024 bytes. Traditionally, system administrators have sought a balance between their need for fast I/O and disk storage availability. Larger block sizes produce fewer I/O requests and thereby produce faster I/O. On average, the last data block allocated to a file is only half full. As the block size increases, so does the amount of wasted disk space. According to David Rusling (1999), most of the advantages associated with a larger block size are obtained via Ext2f's policy of allocating data blocks before they are actually required. This is why it makes sense for the default block size to be 1024 bytes.

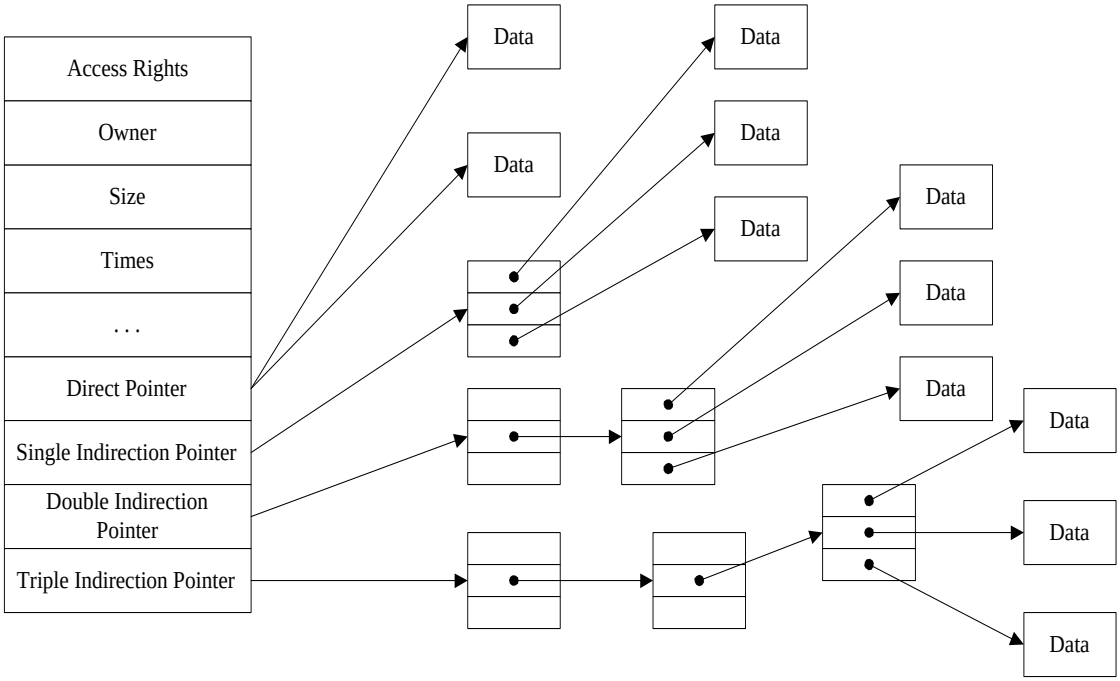
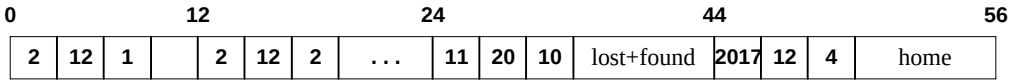


Figure 3. The Second Extended File System inode structure. Source adapted from Beck, M., Bohme, H., Dziadzks, M., Kunitz, U., Magnus, R., & Verworner, D. (1996). Linux Kernel Internals, Reading, MA: Addison Wesley Publishing Company, Inc.

Directories

A directory can contain both files and subdirectories. Each directory is implemented as a special file that contains a list of entries. Each entry consists of a variable length file name paired with an inode number. The system uses the file name to look up the corresponding inode number. The inode number is then used internally to access the file. Since an inode does not contain a field corresponding to the file's name, multiple directory

entries may refer to the same file. From a user's perspective, it is the file name and not its associated inode number that uniquely identifies a file.




```
struct ext2_dir_entry {
        unsigned long inode;                    /* inode number */
        unsigned short rec_len;                 /* length of directory */
        unsigned short name_len;                /* length of filename */
        char    name[EXT_NAME_LEN]              /* filename */
};
```

Figure 4. A Second Extended File System directory entry. Source adapted from Beck, M., Bohme, H., Dziadzks, M., Kunitz, U., Magnus, R., & Verworner, D. (1996). Linux Kernel Internals, Reading, MA: Addison Wesley Publishing Company, Inc.

Every inode maintains a reference counter that records the number of directory entries pointing to that inode. The reference counter is incremented for each active process accessing the file's data blocks. This delays the deletion of a file until all programs accessing its contents have terminated. As each process accessing the file terminates, the reference counter is decremented. When the reference counter reaches zero the file has been deleted. Therefore, the inode and the data blocks allocated to that inode are freed.

Links

Any reference to an inode from a directory is referred to as a hard link. Consider Figure 5 that uses the *ln* command to create a hard link. By specifying *–i* option in

conjunction with the *ls* command, a user can view the inode number associated with each file. After executing the *ln* command, both the file outline and the new file hard-link point to inode 10087. The third element of each directory entry listed in Figure 5 is the link count. The execution of the *ln* command causes the link count for inode 10087 to be incremented.

```
# ls -li

total 164
  10084 -rw-rw-r--   1  root  root   90181        Apr 12 16:21 homework2
  10054 -rw--------  1  root  root   63611        Apr 12 16:42 mbox
   2057 drwx------  2  root  root    1024        Apr 12 00:46 nsmail
  10087 -rw-rw-r--   1  root  root       5        Apr 12 18:48 outline

# ln outline hard-link
# ls -li

total 165
  10084 -rw-rw-r--   1  root  root   90181        Apr 12 16:21 homework2
  10054 -rw--------  1  root  root   63611        Apr 12 16:42 mbox
   2057 drwx------  2  root  root    1024        Apr 12 00:46 nsmail
  10087 -rw-rw-r--   2  root  root       5        Apr 12 18:48 outline
  10087 -rw-rw-r--   2  root  root       5        Apr 12 18:48 hard-link
```

Figure 5. The Second Extended File System hard link.

Every file system is created with a designated number of inodes. Each inode is associated with an inode number that is unique in that file system, but which as a rule is not unique across file systems. This is why hard links cannot be extended across file systems.

In review, files are implemented using three entities. The inode contains information about a file. A directory entry is used to link a file's name to an inode number. The actual contents of the file are stored in file data blocks. These entities serve to separate a file's name, meta-data, and actual data. This separation makes the implementation of hard links possible (Rubini, 1994).

A symbolic link is a file that contains an ASCII string that represents a path name. When the system encounters a symbolic link *L*, it uses the path name stored in *L* to re-access the file system. Linux supports cross-file system symbolic links. Additionally, symbolic links may reference any type of file, including a directory. The principal disadvantage of symbolic links is the overhead associated with their use.

A symbolic link requires an inode and at least one data block, rather than a mere directory entry as in a hard link. Furthermore, the path name to inode conversion takes additional time. Ext2fs mitigates these problems through fast symbolic links. In a fast symbolic link, the path name is not stored in a separate data block. Instead, the path name is stored in the area of the inode typically reserved for the block pointers. Using fast symbolic links eliminates the need for an extra data block, as well as the block transfers needed to access a path name during the symbolic link lookup. Fast links can only be implemented when the target is 60 characters or less. Using the *ln –s* command, creates a symbolic link. The creation of a fast symbolic link is transparent to the user. If possible, the system will create a fast symbolic link.

Notice that the link count for the file outline in Figure 6 does not increase after the creation of soft-link. The directory entry for outline points to inode 10087 that contains the

file's data block pointers. The directory entry soft-link points to inode 6059. However, inode 6059 uses the space reserved for the data block pointers to instead hold a relative path to outline.

```
# ls -li

total 164
 10084 -rw-rw-r--   1 root root 90181       Apr 12 16:21 homework2
 10054 -rw--------  1 root root 63611       Apr 12 16:42 mbox
  2057 drwx------   2 root root  1024       Apr 12 00:46 nsmail
 10087 -rw-rw-r--   1 root root     5       Apr 12 18:48 outline

# ln -s outline soft-link
# ls -li

total 165
 10084 -rw-rw-r--   1 root root 90181       Apr 12 16:21 homework2
 10054 -rw--------  1 root root 63611       Apr 12 16:42 mbox
  2057 drwx------   2 root root  1024       Apr 12 00:46 nsmail
 10087 -rw-rw-r--   2 root root     5       Apr 12 18:48 outline
  6059 -rw-rw-r--   2 root root     5       Apr 12 18:48 soft-link
```

Figure 6. The Second Extended File System soft link.

Device Special Files

Unix allows users to access system character and block devices through device special files. Of particular interest are the block devices on which file systems can reside. These block devices are viewed by Ext2fs as simply a linear collection of blocks. When an I/O request is made on a device file, the request is forwarded to the appropriate device driver. It is the block device driver's responsibility to map such a request to the particular track, sector, and cylinder of its associated disk drive.

Credentials

A unique number, the user id (UID), is associated with each user. UIDs are collected into groups that are identified by a unique group id (GID). Each UID is contained in at least one mandatory group. This group represents the primary group for a specific user. Collectively, these attributes are referred to as user credentials. Credentials are used to set file ownership and access permissions.

A process can be thought of as a running program. Credentials are used to designate execution permissions associated with each process. Every process has an effective UID specifying the user responsible for initiating the process and a real UID designating the user who owns the executable. Additionally, each process has an effective GID specifying the primary group id (PGID) of the user responsible for launching the process and a real GID designating the user who owns the executable.

A process normally executes with the permissions of the effective UID and the effective GID. Information concerning ownership and permissions associated with a file are found in the inode structure. The command *ls –l* as depicted in Figure 7 displays permissions associated with a file. If an executable has the set group id (SUID) bit set, the process executes with the permissions of the real UID. Likewise, if the executable has the set group id (SGID) bit set then the process executes with permissions of the real GID.

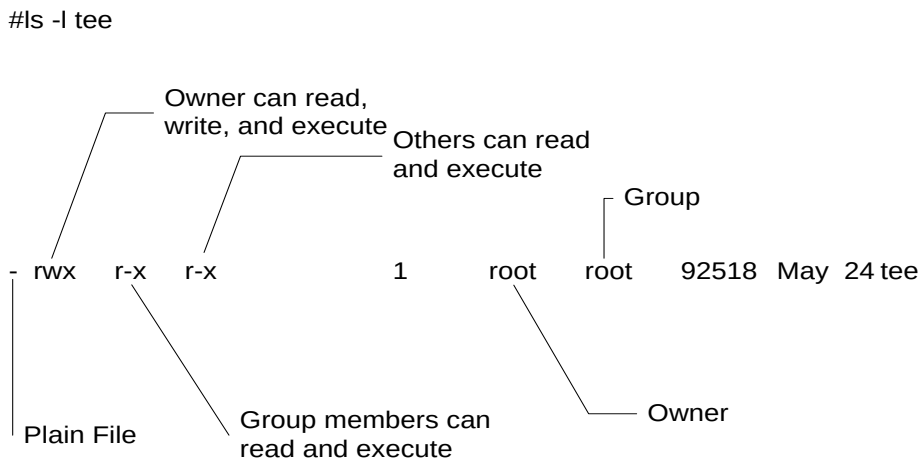


Figure 7. Second Extended File System permissions. Source adapted from Welsh, M. and Kaufman, L. (1995). Running Linux. Sebastopol, CA: O'Reilly & Associates, Inc.

Specific Ext2fs Features

Ext2fs provides a user with the ability to select Berkeley Software Distribution (BSD) or AT&T's System V Release 4 (SRV4) semantics at mount time. If BSD semantics are used, files are created with the same GID as their parent directory. If SRV4 semantics are used, files are also created with the same GID as their parent directory when the parent directory has SGID turned on. However, files in a SRV4 directory that does not have SGID turned on will inherit the GID of the process responsible for their creation.

The superblock is a file system structure that contains vital information about an entire file system. Ext2fs tracks a file system's state through a file system state flag, maintained in the superblock structure. At system boot, each file system is mounted in read-only mode and the file system state flag is tested. If the state flag has a value of not clean then the file system was unmounted improperly.

A few microseconds are required to copy a data block from one memory location to another. A similar disk operation requires several milliseconds. System performance would be severely impeded if users were required to wait for the completion of each disk I/O operation (Vahalia, 1996). Linux offsets this disparity between the speed of disk I/O verses the speed of memory I/O by placing recently accessed data blocks in an area of memory referred to as the buffer cache. The Linux buffer cache utilizes a write-back update policy. If a file is modified the changes are first applied to the buffer cache. Then at intervals ranging from five to thirty seconds the contents of the buffer cache are synchronized with the contents of the disk. If a file system is not unmounted properly there is a possibility that file system modifications have not been synchronized to disk. Therefore, if the file system state flag has a value of not clean at boot then the File System Checker (fsck) is launched to check and repair the file system if necessary.

If the file system state flag has a value of clean then it is re-mounted in read-write mode. Subsequently, the file system state flag is set to not clean to ensure that fsck is executed during the next system boot sequence if the system is shutdown improperly. When a file system is unmounted properly, the file system state-flag is set to clean prior to halting the system. The command *tune2fs –l* can be used to view a file system's superblock. Notice in Figure 8 that the file system state flag is set to not clean because the file system was mounted.

```
# /sbin/tune2fs -l  /dev/hda2
tune2fs 1.04, 16-May-96 for EXT2 FS 0.5b, 95/0/09
Filesystem magic number:          0xEF53
Filesystem revision #:            0
Filesystem state:                 not clean
Errors behavior:                  Continue
Inode count:                      25792
Block count:                      102816
Reserved block count:             5140
Free blocks:                      75450
Free inodes:                      23406
First block:                      1
Block size:                       1024
Fragment size:                    1024
Blocks per group:                 8192
Fragment per group:               8192
Inodes per group:                 1984
Inode blocks per group:           248
Last mount time:                  Sat Apr 12 06:28:05 1997
Last write time:                  Sun Apr 13 02:12:52 1997
Mount count:                      6
Maximum mount count:              20
Last checked:                     Fri Apr 11 19:48:19 1997
Check interval:                   0
Reserved blocks uid:              0 (user root)
Reserved blocks gid:              0 (group root)
```

Figure 8. The Second Extended File System superblock structure.

Ext2fs provides two methods for ensuring that file system checks are conducted at regular intervals. The tune2fs utility can be used to set the maximum mount count field found in the superblock structure. This field is used in conjunction with the mount count field to determine if it is time to run fsck. Each time a file system is mounted, mount count is incremented. When mount count equals maximum mount count the file system is checked by fsck. Tune2fs can also be used to specify a value for the superblock's check

interval field. The check interval field is used in conjunction with the superblock's last checked field to enforce time based file system checks. Setting check interval to zero disables time based file system checking.

Ext2fs Physical Structure

The Ext2fs physical structure is based on the Berkeley Fast File System (FFS) and is designed to address limitations of the System V File System (s5fs). A major concern with s5fs is that no redundant copies of the superblock are maintained (Vahalia, 1996). Therefore, if an s5fs superblock is corrupted the file system becomes unusable. Figure 9 depicts the physical structure of s5fs.

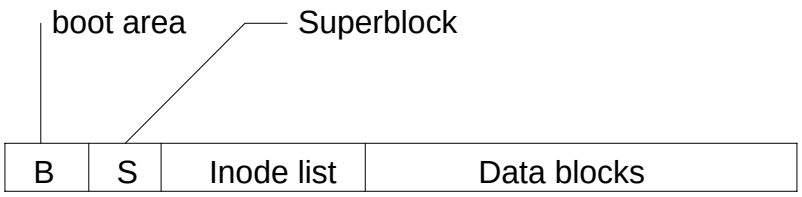


Figure 9. The System V File System physical structure. Source adapted from Vahalia, U. (1996). Unix Internals. Upper Saddle River, NJ: Prentice Hall.

As depicted in Figure 9, all s5fs inodes are grouped at the beginning of the file system. This single group of inodes is followed by a single group of data blocks. Accessing a file requires reading the appropriate inode followed by the file data. Such segregation increases the average read-write head seek distance. Since seeking is the most

costly I/O operation, a significant reduction in I/O performance results. It also makes the time required for an I/O operation less predictable.

Ext2fs maintains redundant copies of critical file system structures. This is done through block groups as depicted in Figure 10. Each block group contains six types of structures: a superblock, group descriptors, a block bitmap, an inode bitmap, an inode table, and data blocks. A description of the form, content, and purpose of each structure follows.

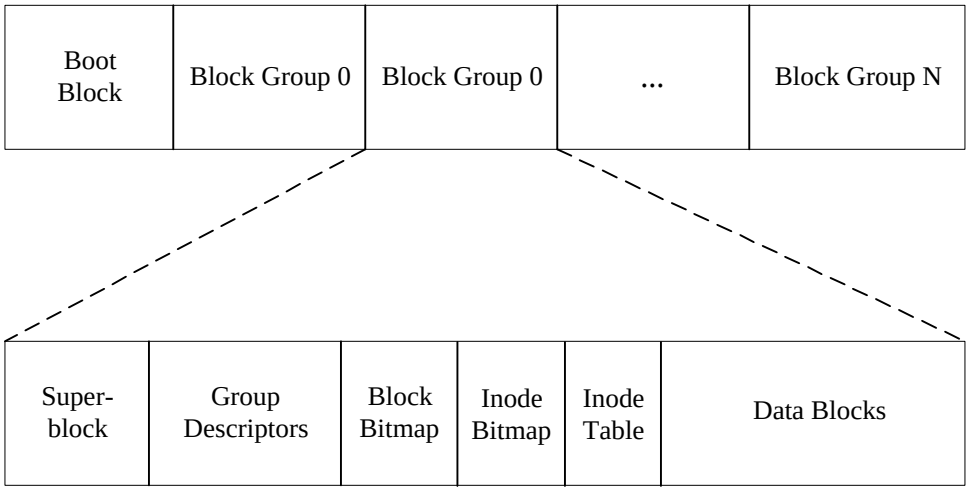


Figure 10. The Second Extended File System physical structure. Source adapted from Beck, M., Bohme, H., Dziadzks, M., Kunitz, U., Magnus, R., & Verworner, D. (1996). Linux Kernel Internals, Reading, MA: Addison Wesley Publishing Company, Inc.

Superblock

A partition table is used to describe how a disk can be divided into up to four disk-like entities referred to as primary physical partitions. If further subdivision is necessary, a physical partition is used as an extended partition. The extended partition can then be

subdivided into several logical partitions. For any hard disk to be usable, at least one partition must exist. The superblock structure anchors Ext2fs to a disk partition (Bentson, 1996). The superblock contains file system meta-data and control information, such as the number of inodes in the file system and per block group.

Group Descriptor

A copy of all file system group descriptors is maintained in each block group. A group descriptor contains meta-data pertaining to a block group such as the block numbers associated with the block and inode bitmaps. Additional information found in each group descriptor includes the block group's free block count, free inode count and used directory count.

Block Bitmap

The block bitmap is a single block used to map the entire set of data blocks allocated to the block group. Each bit of the block bitmap indicates the allocation status of a specific data block from the block group. A value of one indicates that the block has been allocated, a value of zero indicates otherwise. Bit zero of byte zero in the block bitmap represents data block zero. Bit zero of byte one represents data block eight and so on. If the block size is 1024 bytes, then 8 x 1024 = 8192 blocks can be mapped per group.

Inode Bitmap

The format of the inode bitmap is the same as that of the block bitmap. The number of inodes is usually significantly less than the number of data blocks. Therefore, only a portion of each inode bitmap is typically used.

Each file has a single inode associated with it. By default Ext2fs reserves one inode for every 4096 bytes of file system space. It is possible to run out of inodes and still have file system space available. If such a situation occurs then existing files can be enlarged but no new files can be created.

Inode Table

Inodes are grouped into tables. Each table is placed in a separate block group. The goal is to place inodes and their related data blocks in close proximity to one another (Oxman, 1995).

CHAPTER 4

SMART DISKS

One of the most important performance measures associated with a typical multi-user computer system is response time. Performance of the I/O subsystem imposes a lower bound on a computer's responsiveness. The slowest component of the I/O subsystem, other than the end-user or the printer, is the hard disk. On a Unix system, the best way to improve I/O performance is through use of an effective buffer cache (Bentson, 1996). Such a buffer cache is maintained in main memory by the operating system and designed to minimize the number of disk accesses. Given an efficient buffer cache, one must then focus on increasing the performance of the disk drive.

Adaptive disk research focuses on copying or moving frequently accessed data to a pre-specified location on the disk drive. The term adaptive, as used above, means that no static knowledge as to the frequency with which data will be accessed is required. All adaptive algorithms require that the read and write request streams be monitored for the block device in question. The intent is to reduce disk overhead due to seek operations. If frequently accessed data is placed at a common location, the read-write heads will tend to dwell near that location. As the absolute difference between average seek time and average rotational latency continues to shrink, the importance of adaptive algorithms will increase. The most visible difference between the various studies is the granularity at which data is copied or moved. This section will primarily discuss an adaptive technique developed by Sedat Akyurek and Kenneth Salem (1995) referred to as Adaptive Block Rearrangement. Adaptive Block Rearrangement focuses on reorganization of data at the granularity of

blocks. Initially, other studies focusing on the movement of cylinders and files will be considered briefly.

Organ Pipe Heuristic

The organ pipe heuristic places data optimally if data references are derived from an independent random process with a known fixed distribution. The organ pipe heuristic is used in adaptive disk management in the following manner. The most frequently accessed data is placed on the middle cylinders of the disk. The heuristic then places the next most frequently accessed data to either side of the middle cylinders. This process continues until the outer and inner tracks have been reached. Problems with the organ pipe heuristic are that data references are not drawn from a known fixed distribution and they are not independent (Akyurek & Salem, 1995). As discussed in previous sections of this document, data references tend to exhibit locality over time and space. Variations of the organ pipe heuristic using more realistic reference patterns have demonstrated limited success.

Cylinder Shuffling

The cylinder-shuffling algorithm presented by Vongsathorn and Carsion (1990) reorganizes cylinders based on a measured frequency of access. The algorithm assigns a number from zero to ($n - 1$) where n is an integer identifying the total number of cylinders per disk platter. The frequency of disk cylinder references is then monitored over a period of time. Reference counts are analyzed, the Organ Pipe heuristic is applied, and a permutation of the set of all cylinders is constructed. The study reported seek time

reductions of 40 to 45 percent. For test results to be meaningful, the test must be applied to varying workloads (Rummler & Wilkes, 1994). Their study did not report on seek time reductions but did note disk service time reductions of up to 10 percent.

The problem with cylinder shuffling is the overhead required to re-shuffle the cylinders. As stated by Akyurek and Salem (1995), it is also impossible for this strategy to eliminate seeks. By definition a seek requires the read-write head to move to the track on which the requested data resides. Cylinder shuffling may lessen the required seek distance and thereby hopefully the seek time. However, data is not being moved or copied from one track to another and a seek is still required.

The iPcress File System

The iPcress File System monitors references to files and assigns a temperature to each file. The term temperature, as used above, is defined as frequency of access divided by file size. Files with high temperatures are moved to the center of the disk. A problem with iPcress is that it fails to relocate the file system meta-data. As stated by Akyurek and Salem (1995), this approach requires file system modifications.

Implementation

Adaptive placement strategies may be implemented in file systems, disk drivers, disk controllers, or in some combination of the three (Akyurek & Salem, 1995). File system level implementations are difficult to port, being file system specific, and often operating system specific. Device driver algorithms are more readily portable. Controller-level implementations are operating system non-specific and can make detailed assumptions

about device hardware characteristics. Unfortunately, controller-level algorithms are also costly to implement and therefore beyond the scope of this project.

Copying Verses Moving

There are several advantages associated with copying hot data to a reserved area on the disk. Copying does not undermine file system specific placement optimization. If a unit of data cools (is referenced less often) only modified data must be written back to its original location. Since there will be two copies of the frequently accessed data unit, the original copy could potentially be used as well. As discussed in chapter 3, disk storage densities have increased dramatically over the past few years. Studies have shown that only a small percentage of the disk's data needs to be duplicated to obtain significant seek distance and seek time reductions.

Granularity

As previously mentioned, a major distinction among the different algorithms is the granularity of the data to be moved or copied. Large granularity provides less control. If a large granularity is used some non-frequently accessed data will ultimately be moved to the reserved region of the disk inadvertently. Consider the pervious discussion on cylinder shuffling. Once shuffling has completed, the cylinder located in the center of the disk will still contain some non-frequently accessed data.

Overhead associated with monitoring block references increases as the granularity decreases. Adaptive Block Rearrangement copies a small number of frequently referenced data blocks to a reserved area located in the center of the disk. By rearranging a small

percentage of the disk's data blocks Akyurek and Salem (1995) proved that, dependent on the workload, 30 to 85 percent reductions in seek times were achievable. A number of block reference counters, no less than the number of reserved blocks, must be maintained in main memory. Test results presented by Akyurek and Salem (1995) demonstrated that maintaining more reference counters than the number of blocks reserved for rearrangement contributed little to a system's performance. Over the last few years, the cost of computer memory has dropped sharply. The previous observation and data presented by Akyurek and Salem (1995) indicates that the smaller block granularity is the best choice.

Adaptive Block Rearrangement

Akyurek and Salem (1995) implemented Adaptive Block Rearrangement at the device driver level. In addition to the device driver, two user level processes were used. The Reference Stream Analyzer is a user level process responsible for monitoring the stream of data requests. The device driver maintains an area in main memory accessible to user level processes. When the device driver receives a read-write request it writes the associated block number to this area of main memory. The Reference Stream Analyzer reads the block number, and updates or creates the appropriate reference counter. Periodically, reports are generated by the Reference Stream Analyzer listing hot block order by frequency of reference. The Block Arranger then uses the Reference Stream Analyzer's report to determine which blocks should be copied to the reserved portion of the disk.

The Hot Block Table lists those blocks that have been copied to the reserved portion of the disk. Each entry in the table contains a block number and a pointer that maps a data

block onto the reserved portion of the disk. The Block Arranger copies each block found in the Reference Stream Analyzer's report to the reserved portion of the disk and updates the Hot Block Table.

The device driver is responsible for hiding the reserved portion of the disk from the rest of the operating system. From the operating system's perspective, the disk will appear smaller than it actually is. Upon receiving a read or write request, the device driver first checks the Hot Block Table for the relevant block number. If the block number is found in the Hot Block Table, the request is mapped onto the reserved area of the disk.

Testing Conditions

Akyurek and Salem (1995) used a simulator as their primary testing tool. Prior to testing, Adaptive Block Rearrangement was implemented for approximately a year on a Sun SPARC station Sakarya-2. Data obtained from the implementation were used to validate the simulator. Traces for the simulator were obtained from three different UNIX systems: Ballast, Sakarya-1, and Snake. Ballast and Sakarya-1 are Sun SPARC stations. Snake is a Hewlett Packard 9000/720 workstation. Both Ballast and Sakarya-1 were use as Network File System (NFS) servers. During the latter half of the testing period, Sakarya-2 was converted to a stand-alone system. Snake acted as a file server for the faculty, staff and students of the University of California, Berkeley (Akyurek & Salem, 1995). Both Sakarya-1 and Ballast used a disk drive that provides support for read-ahead buffering. The disk drive on Snake had a feature known as immediate write reporting. This features causes the disk to report a write operation as being completed once the data to be written was transferred from the host to the disk controller's buffer (Akyurek & Salem, 1995).

Immediate write reporting is active only if the write operation is a physical extension of the previous write operations (Akyurek & Salem, 1995).

Reference Stream Analysis

Due to the limited number of reference counters, the Reference Stream Analyzer will make mistakes. When considering reference analysis there are two types of errors. Let $\tilde{a}(i)$ be the list of hot blocks produced after the ith monitor period. A perfect monitor maintains a reference counter for every data block on the disk drive being monitored excluding blocks reserved for hot data. Let $a(i)$ be the list of hot blocks produced by a perfect monitor after the ith period. If a reference counter is maintained for each disk block then $\tilde{a}(i) = a(i)$. Any difference between $\tilde{a}(i)$ and $a(i)$ is due to the limited number of reference counters (Akyurek & Salem, 1995). The second type of error results from temporal variations in reference patterns. Temporal variation refers to the Reference Stream Analyzer's ability to accurately predict the hot blocks for the next period or, more formally, the difference between $a(i)$ and $a(i + 1)$. Reference counters exceeding the number of reserved blocks did not significantly improve the Reference Stream Analyzer's ability to accurately predict hot blocks (Akyurek & Salem, 1995).

A review of the Sakarya-2 traces showed higher temporal variation than in the other system's traces. This is why Sakarya-2 performed poorly in comparison. The performance of Adaptive Block Rearrangement depends on file system changes occurring slowly over a period. Consider the NFS file servers Ballast and Sakarya-1. NFS file systems are often mounted read-only. Therefore, only disk blocks that contain file system meta-data are modified. This serves to isolate those disk blocks that change slowly over a period of time.

Data Placement

The two data placement techniques used to organize hot blocks in the reserved region of the disk were the organ pipe heuristic discussed earlier, and serial placement which positions the blocks sequentially based on their addresses. Test results indicated that the data placement policy had little effect on the performance of Adaptive Block Rearrangement. In those cases where Adaptive Block Rearrangement performed best, the organ pipe, heuristic slightly outperformed serial placement. As pointed out by Akyurek and Salem (1995), this was due to the creation of artificial hot spots in the reserved region of the disk.

Head Scheduling

The head scheduling algorithms tested in conjunction with Adaptive Block Rearrangement included FCFS, SCAN, and CSCAN. Since scheduling algorithms with a task queue length of one degrade to FCFS, FCFS was the base case against which the other algorithms were tested. The benefits obtained from any of the head scheduling algorithms were minimal. This is because no head scheduling algorithms incorporating rotational optimization were tested. Adaptive Block Rearrangement causes the read-write heads to linger over a reserved region of the disk. This lingering limits the effectiveness of any head-scheduling algorithm that focuses purely on seek optimizations. For a head-scheduling algorithm to be effective, a sizeable disk queue is necessary. The mean disk queue lengths on the machines tested ranged from 0.5 to 2.3 requests. Due to small mean queue lengths, it is questionable if the head-scheduling algorithms would have any impact even on systems not implementing Adaptive Block Rearrangement.

Conclusions

The success of any adaptive disk algorithm is dependent on the ability to accurately predict future disk activity. If file system changes occur slowly over time then an adaptive algorithm will be more likely to correctly predict future disk activity. Due to recent increased disk storage densities, it is better to copy rather than move hot data. Copying the hot data does not destroy file system specific placement optimizations. In selecting the granularity of the data to be copied, it is better to choose a finer granularity. A finer granularity provides greater control over what data is placed in the reserved portion of the disk. The actual impact of head scheduling on adaptive block management remains unclear. Further testing needs to be conducted with traces that have larger mean queue lengths than those used by Akyurek and Salem (1995).

C H A P T E R 5

PERFORMANCE TEST RESULTS

System design and performance tuning significantly impact system performance. In particular, the use of DMA, even on inexpensive IDE disk drives significantly reduces processor time consumed in performing disk operations and increases drive throughput. The performance tests presented below measure the impact of PCI bus mastering DMA on system responsiveness and drive throughput. Each performance test was executed under PIO mode 4, DMA multiword 3, and Ultra-Wide SCSI. DMA test results are compared to PIO results to evaluate performance gains associated with PCI bus mastering DMA. Another strategy for improving I/O subsystem performance uses high-performance SCSI drives. Unfortunately, SCSI hardware is often too expensive to use on low-end multi-user system. SCSI test results are compared to IDE DMA results to provide a basis for future cost-benefit analysis.

Test Specifications

Performance test results only have value if they are reproducible. It is the experimenter's responsibility to record information necessary to reproduce performance test results. Accordingly, this chapter will first specify the hardware and operating system platforms on which performance tests were conducted. For each benchmark utility used, a general discussion will be presented followed by a detailed explanation describing how the utility was executed and the results obtained.

Hardware Specifications

All tests were executed on a Concerto III+ MS-6204 Micro NLX system. Critical system hardware specifications are listed in Figure 16, Appendix B.

Operating System Specifications

Initially, the Maxtor 92048U8 and Western Digital WDE1830 drives were partitioned as depicted in Figure 11. A popular Linux distribution, Red Hat 6.1, was then installed on each system configuration (see Figure 16). The Linux 2.2.12 kernel image included with the Red Hat 6.1 distribution was used for all testing. All performance tests were executed on the test partition depicted in Figure 11. Prior to each performance test, the test partition was reformatted to eliminate file system factors such as disk fragmentation.

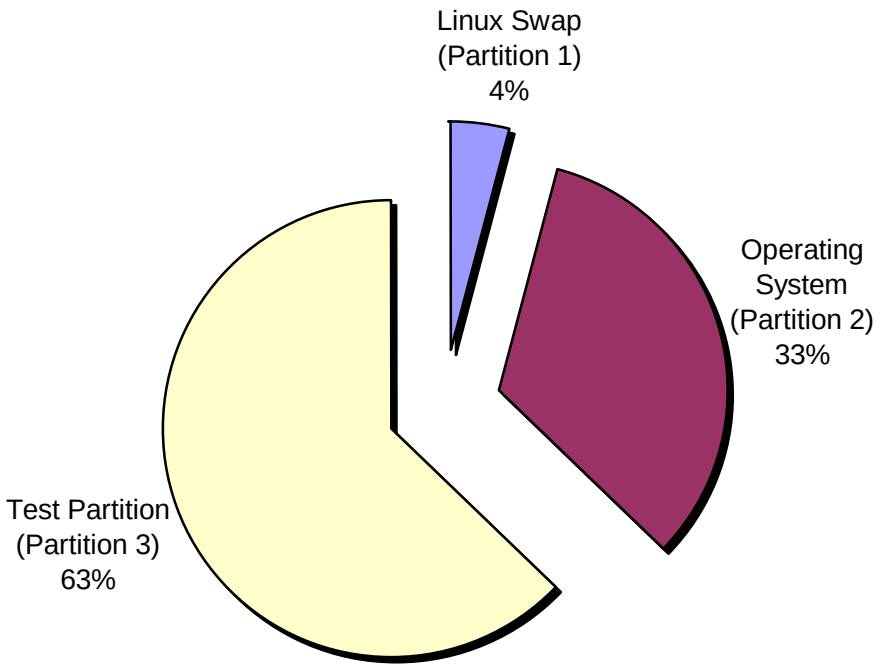


Figure 11. Drive partition scheme used for performance tests.

Testing Procedure

The IDE data transfer protocols were tested first. Each performance test was initially executed under PIO mode 4 and then under DMA multiword 3. On completion of IDE performance testing, the Maxtor drive was removed from the test system. The Western Digital SCSI drive and the Adaptec 2940UW SCSI host adapter were then installed on the system. Each performance test was then executed on the SCSI system configuration.

Prior to each experiment, the test partition was formatted and necessary benchmark utilities were copied to the test partition. The system was then rebooted to ensure consistency of the memory subsystem. On reboot, the experimenter logged into the system and changed to the appropriate directory on the test partition. If required, the IDE drive transfer mode was set. The appropriate performance test was then initiated. On test completion, results were carefully recorded.

Drive Performance Test Results

Bonnie

Bonnie is a simple yet powerful benchmark utility written by Tim Bray (1996). This utility is designed to identify I/O bottlenecks and determine raw performance gains associated with individual optimizations. Bonnie performs a series of synchronous sequential write, sequential read, and random read operations on a file of specified size. The user should specify a file size several times the size of available main memory; otherwise, Bonnie ends up testing the performance of an operating system's caching algorithm (Bray, 1996). Since the system under test had 256 MB of main memory (see

Figure 16), a file size of 1024 megabytes was selected for all Bonnie runs. For each performance test, Bonnie reports the bytes processed per elapsed second and the percentage of CPU time required executing the file operations.

When benchmarking multiple computer systems with different operating systems and different hardware components Bonnie fails to adequately measure disk performance. Other computer system components such as system libraries, memory latency, and system call latency cloud benchmark results making it difficult to determine the effectiveness of individual optimizations. Because all performance tests presented in this thesis were executed on the same system, one can use Bonnie to accurately determine the effectiveness of individual hard drive optimizations.

When writing data on a character basis DMA multiword 3 incurs more CPU overhead than PIO mode 4. Bonnie character-write results listed in Table 5 show a lower CPU overhead was incurred under PIO mode 4 than under DMA multiword 3 or Ultra-Wide SCSI. The higher CPU overhead incurred under DMA multiword 3 and Ultra-Wide SCSI is due to the setup costs and higher maximum transfer rates. The higher maximum transfer rate provided by DMA multiword 3 and Ultra-Wide SCSI is reflected by the average kilobytes per second listed in Table 5.

Table 5

Bonnie Character Write Results (Average of Five Trials)

Transfer Protocol	KB/Second	CPU Percentage
PIO Mode 4	2383	90.62
DMA Multiword 3	6561	96.16
Ultra-Wide SCSI	6624	96.98

Note. PIO = Programmed Input/Output, DMA = Direct Memory Access, SCSI = Small Computer Systems Interface, KB = kilobyte.

When writing data on a block basis DMA multiword 3 and Ultra-Wide SCSI outperformed PIO mode 4. Bonnie block-write results listed in Table 6 show a DMA multiword 3 data transfer rate approximately five times that of PIO mode 4. The CPU overhead associated with PIO Mode 4 is approximately four times the CPU overhead associated with DMA multiword 3. CPU overhead incurred when using PIO mode 4 is less when writing data using efficient block writes than when writing data on a character basis. This is due to overhead associated with the large number of kernel-level putc() macro invocations required to write a 1024 MB file. Results depicted in Table 6 demonstrate the efficiency of block-writes when writing large files.

Due to virtual addressing, memory appears contiguous to a user space process. In reality, a program's address space is fragmented throughout main memory. Therefore, writing a large file sequentially to disk often requires that a single write operation be broken up into several smaller transfers. Scatter/gather is a technology that allows for the

specification of multiple data source addresses in a single SCSI command. This explains the higher throughput obtained under Ultra-Wide SCSI.

Table 6

Bonnie Block Write Results (Average of Five Trials)

Transfer Protocol	KB/Second	CPU Percentage
PIO Mode 4	3627	73.86
DMA Multiword 3	18104	19.58
Ultra-Wide SCSI	19099	21.28

Note. PIO = Programmed Input/Output, DMA = Direct Memory Access, SCSI = Small Computer Systems Interface, KB = kilobyte.

By default, the Maxtor 92048UA drive's internal buffer cache is only used for read caching. Write caching can be enabled via the hdparm IDE drive utility include with most Linux distributions. However, write caching was not enabled for any of the Bonnie runs presented in this thesis.

DMA multiword 3 more efficiently utilizes the drive's buffer cache than does PIO mode 4. When using DMA multiword 3, a request is issued and data is then transferred directly from the buffer cache to main memory. This is advantageous when reading large files in a sequential manner. When dealing with large sequentially accessed files, PIO mode 4 fails to take full advantage of the drive's cache. The drive must wait for the CPU to transfer data to main memory and then issue instructions to transfer the next word of

data. Bonnie block-read results listed in Table 7 show a DMA multiword 3 data transfer rate approximately seven times that of PIO mode 4.

Tagged Command Queuing (TCQ) refers to the ability of a SCSI drive to queue multiple outstanding commands. TCQ allows the SCSI drive-controller to apply seek and rotational optimizations to pending requests on the command queue. Conversely, each IDE controller has a single data channel that supports a single pending request.

The performance tests presented in this thesis use synchronous I/O. This means a benchmark utility will block until a requested I/O operation has completed. Therefore, the Western Digital SCSI drive does not benefit from TCQ. This explains the higher transfer rate associated with the Maxtor IDE drive when executing under DMA multiword 3.

Table 7

Bonnie Block Read Results (Average of Five Trials)

Transfer Protocol	KB/Second	CPU Percentage
PIO Mode 4	2797	95.40
DMA Multiword 3	20214	11.4
Ultra-Wide SCSI	17714	11

Note. PIO = Programmed Input/Output, DMA = Direct Memory Access, SCSI = Small Computer Systems Interface, KB = kilobyte.

Prior to beginning random read testing, Bonnie forks four child processes. Each process executes 4000 seek operations to random locations in the 1024 MB file. On 400 out of the 4000 seek operations, each child process modifies and rewrites the requested

block. PIO mode 4, DMA multiword 3, and Ultra-Wide SCSI random read test results are listed Table 8. Due to overhead associated with seek operations, the percentage of CPU time associated with both PIO mode 4 and DMA multiword 3 is low. Instead of transferring data, the CPU is executing other tasks while seek operations are taking place.

Disk drive read and write request are reordered by Linux in the operating system cache to reduce unnecessary seek operations. This seek reordering allows DMA multiword 3 to use the Maxtor drives internal buffer cache more efficiently than PIO mode 4. This explains why the disparity between required CPU times using PIO mode 4 verses DMA multiword 3 has increased.

Notice the higher seek rate produce under the Ultra-Wide SCSI configuration. This higher seek rate is primarily due to the superior components normally found on SCSI drives. Consider the Western Digital and Maxtor drive specifications listed in Figure 16. The Western Digital specifications list an average seek time that is 2.1 ms less than the average seek time listed in the Maxtor specifications.

Table 8

Bonnie Random Read Results (Average of Five Trials)

Transfer Protocol	Seeks/Second	CPU Percentage
PIO Mode 4	84	25.98
DMA Multiword 3	123	2.14
Ultra-Wide SCSI	133	2.48

Note. PIO = Programmed Input/Output, DMA = Direct Memory Access, SCSI = Small Computer Systems Interface.

Hdparm

Hdparm is an IDE performance utility written by Mark Lord (1994) that is included with all major Linux distributions. This utility allows users to tune numerous IDE disk drive parameters. Hdparm can also be used to perform timings of cache and disk drive reads for benchmark and comparison purposes. When executed with the *–t* option hdparm will evaluate drive performance when reading through the Linux buffer cache with no prior caching of data (Lord, 1994). More simply, hdparm displays how fast a drive can sustain sequential reads under the Linux operating system with no file system overhead. When executed with the *–T* option hdparm measures throughput associated with a system's processor, cache, and memory (Lord, 1994). When executed with both of the above options, hdparm applies the *–T* option results as a corrective factor to the *–t* results (Lord 1994). Since this thesis is concerned with disk performance rather than efficiency of the Linux buffer cache algorithm, only timings for disk drive reads are displayed.

For accurate results, Lord (1994) recommends that the above timings be repeated a minimum of two to three times. The hdparm test results depicted in Table 9 are based on the average values obtained from five consecutive runs of hdparm using PIO mode 4, DMA multiword 3, and Ultra-Wide SCSI. The disk drive reads are performed on a 64 MB file using efficient block reads.

Bonnie test results listed in Table 7 show that PIO mode 4 fails to take full advantage of the Maxtor drive's internal buffer cache. Hdparm benchmark results listed in Table 9 support this assertion. It took approximately six times longer to read a 64 MB file using PIO mode 4 verses DMA multiword 3. As indicated by Bonnie block-read test results

listed in Table 7, hdparm results show that the Maxtor drive when operating under DMA multiword 3 provided a higher data transfer rate than the Western Digital SCSI drive.

Table 9

Hdparm Results (Average of Five Trials)

Transfer Protocol	MB/Second	Seconds Required
PIO Mode 4	3.48	18.3
DMA Multiword 3	20.48	3.11
Ultra-Wide SCSI	18.79	3.41

Note. PIO = Programmed Input/Output, DMA = Direct Memory Access, SCSI = Small Computer Systems Interface, MB = megabytes.

IOzone

CPU utilization and hard drive throughput are critical to the performance of a multi-user disk I/O bound system. IOzone is a file I/O benchmark utility maintained by Don Capps based on original work done by Will Norcott (1991) that generates and measures a variety of file operations. To simulate a multi-user disk I/O bound system IOzone was executed in multi-process mode launching one to eight child processes each performing synchronous file I/O in 64 KB transfer sizes on a 1024 MB file. Only IOzone sequential read and write results are presented in this section.

The IOzone results depicted in Figures 12 and 13 support assertions that the Maxtor drive's buffer cache is not effectively utilized when the drive is operating in PIO mode 4. IOzone benchmark tests for both read and write operations produced an average transfer

rate in the vicinity of 400-kilobytes per second when executing PIO mode 4 transfers. PIO mode 4 test results show that the number of IOzone processes had little impact on the average transfer rate. This is largely due to CPU overhead incurred when executing PIO mode 4 transfers. The average throughput when executing in PIO mode 4 was slightly higher for write testing as opposed to read testing. Due to efficient utilization of the Maxtor drives buffer cache, the average throughput when executing DMA multiword 3 transfers is significantly higher for read testing as opposed to write testing.

As the number of writing processes increases, the data transfer rate associated with Ultra-Wide SCSI gradually decreases. The SCSI protocol incurs a higher setup overhead than DMA. However, this is typically offset thought the use of asynchronous I/O.

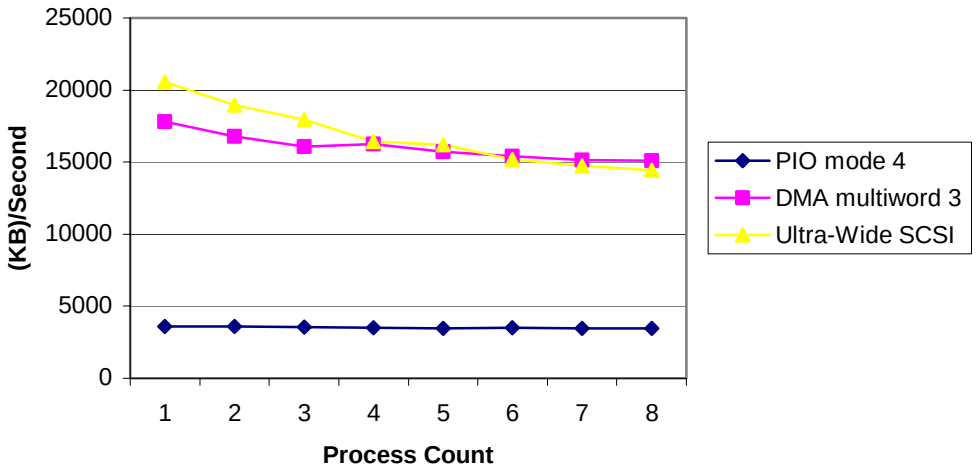


Figure 12. IOzone multi-process write results.

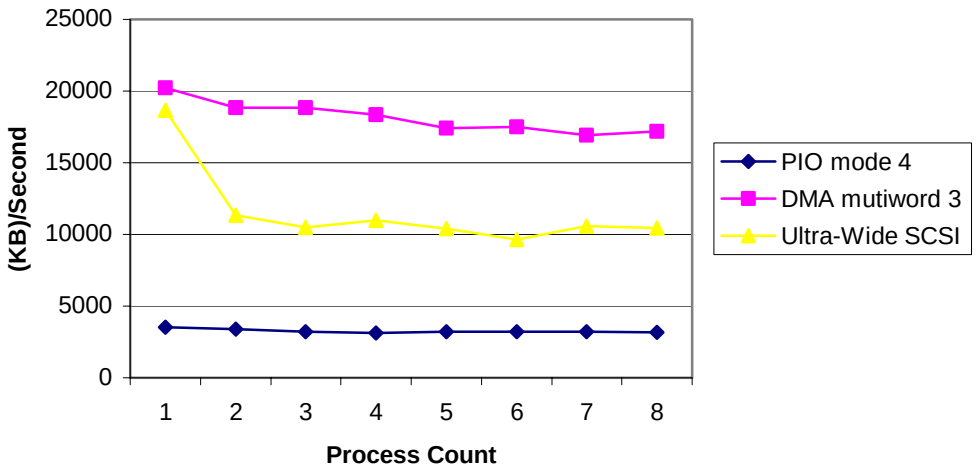


Figure 13. IOzone multi-process read results.

Conclusions

Performance test results presented in this section show that PCI bus mastering DMA when used properly, improves the responsiveness and throughput of IDE drives by as much as a factor of seven. The magnitude of this improvement shows the importance of operating system support for PCI bus mastering DMA in low-end multi user systems. Currently, many operating systems such as Linux and Windows 98 do not utilize DMA by default. Instead, the end user or system administrator must activate DMA.

To efficiently optimize an individual computer system one must consider the usage patterns associated with the system. Due to time constraints, no asynchronous I/O performance tests were conducted. This limitation might cause one to incorrectly draw the conclusion that the high cost associated with high-performance SCSI drives is unwarrantable. However, consider a system on which the majority of disk I/O activity is

composed of asynchronous write operations. Such a system would benefit greatly from SCSI technology such as scatter/gather provided the host operating system supports this technology.

CHAPTER 6

SUMMARY

Strategies for improving I/O subsystem performance presented in this thesis were disk-head scheduling, adaptive disk rearrangement, and DMA. This thesis focuses primarily on DMA, which offers the most potential gain while requiring the least amount of work. Reducing the CPU overhead associated with disk I/O is critical to multi-user system responsiveness. DMA allows data to be sent directly from an attached device to a computer system's main memory and thereby reduces CPU overhead. PCI bus mastering allows modern IDE disk controllers to manipulate main memory without utilizing slower motherboard-resident DMA controllers. The lower CPU overhead associated with PCI bus mastering DMA, allows the IDE interface to use more of a drive's potential sustained transfer rate.

Modern IDE disk drives, motherboard-chipsets, and operating systems provide PCI bus mastering support. However, a typical multi-user operating system must function properly on numerous hardware configurations. Therefore, many operating system manufacturers choose safe but less than optimal default system settings. Operating systems such as Windows 98 and Red Hat Linux do not by default exploit PCI bus mastering DMA. The procedure to enable operating system support for PCI bus mastering DMA is usually straightforward. Nevertheless, many system administrators are unaware of the substantial performance gains associated PCI bus mastering DMA. Conventional wisdom holds that I/O subsystem performance can only be substantially improved through the use of costly high-performance SCSI disk drives (Merkey, 1999). Yet, drive performance test results

presented in chapter 5 showed that PCI bus master DMA outperforms Ultra-Wide SCSI on a single drive system when using synchronous I/O. This thesis will help disprove the commonly held perception that SCSI drives always outperform IDE drives (Merkey, 1999). The remainder of this chapter will discuss some of the most important ideas and concepts that were learned in the development of this thesis. Each topic covered includes suggestions for future research.

Interface Performance Analysis

The drive performance tests presented in chapter 5 were designed to measure the impact of PCI bus mastering DMA on IDE performance, relative to PIO and SCSI performance. Test results showed that PCI bus mastering DMA when used properly improves the responsiveness and throughput of IDE drives by as much as a factor of seven. Due to time constraints, only synchronous I/O performance tests were conducted. This unfortunate limitation can cause one to mistakenly draw the conclusion that the high cost associated with high-performance SCSI drives is always unjustifiable. It is important to understand that computer optimization is system specific. To efficiently optimize an individual computer system one must consider the system's usage patterns, as well as, the capabilities and limitations associated with the host operating system. Consider a system on which the majority of disk activity is composed of asynchronous write operations. This system would benefit greatly from SCSI technology such as scatter/gather and request reordering provided the host operating system supports this technology. Future research should be conducted to measure the performance of IDE PCI bus mastering relative to SCSI using asynchronous I/O on a single drive system.

Adaptive Block Rearrangement

As discussed in chapter 4, Adaptive Block Rearrangement (Akyurek & Salem, 1995) attempts to reduce seek times by copying frequently accessed (hot) data to a reserved location on the disk drive. Akyurek and Salem (1995) placed the hot data disk cache in the center of the disk. This disk cache placement strategy assumes that every track on disk contains the same amount of data, an assumption that was true of standard disk drive technology of the early 1990's. Disk drive technology has changed since Akyurek and Salem's original work. Multi Zoned Recording (MZR) (Quantum Corporation, 2000), a recent technological innovation, allows the outer tracks of a disk to contain more data than the inner. Because more data is stored on the outer tracks, these tracks offer a higher throughput. Due to the widespread use of MZR in today's modern disk drives, the optimal location for the hot data cache may have moved from the center to the outer tracks of the disk. Future research should be conducted to determine if the optimal location in which to copy hot data has moved from the center of the disk. If the optimal hot data location has moved, this research should determine the new optimal disk location for hot data.

Adaptive Block Rearrangement and Disk-head Scheduling

Akyurek and Salem (1995) considered several disk-scheduling algorithms to determine their impact on Adaptive Block Rearrangement. It was determined that the head scheduling algorithms did not significantly influence the performance of Adaptive Block Rearrangement. The above analysis is incomplete. Only head-scheduling algorithms that focus solely on seek optimization as the means for improving disk performance were considered. Adaptive Disk Rearrangement places hot data in a central location on the disk

drive thereby causing the read-write heads to dwell near this location. Consequently, it is not surprising that the disk-scheduling algorithms considered had little impact on the performance of Adaptive Disk Rearrangement. As note in chapter 3, the absolute difference between average seek time and average rotational latency is decreasing. Further research is required to accurately evaluate the impact of head-scheduling algorithms on Adaptive Disk Rearrangement. This research must consider head-scheduling algorithms that include seek and rotational optimizations. Mean task queue lengths produced by the traces gathered for testing were between 0.5 and 2.3 requests. In general, as the mean queue length decreases so does the effectiveness of head scheduling. The traces used for future research should be obtained from heavily utilized I/O bound multi-user systems. Such systems will produce more realistic (larger) mean task queue lengths.

Conclusions

There are many strategies for improving I/O subsystem performance. Of the strategies addressed, DMA offers the highest cost-benefit ratio. The bottom line is that PCI bus mastering DMA improves IDE drive performance by as much as a factor of seven and outperforms Ultra-Wide SCSI on single drive multi-user systems when using synchronous I/O. However, additional research is necessary to evaluate the performance of IDE PCI bus mastering DMA relative to SCSI using asynchronous I/O.

BIBLIOGRAPHY

BIBLIOGRAPHY

Akyurek, S. & Salem, K. (1995). Adaptive Block Rearrangement. ACM Transactions on Computer Systems, 13(2), 89-121.

Beck, M., Bohme, H., Dziadzks, M., Kunitz, U., Magnus, R., & Verworner, D. (1996). Linux Kernel Internals, Reading, MA: Addison Wesley Publishing Company, Inc.

Bentson, R. (1996). Inside Linux, Seattle, WA: Specialized Systems Consultants, Inc.

Bray, T. (1996). Bonnie [Computer software]. Vancouver, BC, Canada.

Card, R., Ts'o, T., & Tweedie, S. (1994). Design and Implementation of the Second Extended Filesystem. Proceedings of the First Dutch International Symposium on Linux, ISBN 90-367-0385-9.

Kozierok, C. M. (2000). The PC Guide. Site Version 1.11.0. Unpublished manuscript. Retrieved February 7, 2000 from the World Wide Web: http://www.pcguide.com/topic.html.

Lord, M. (1994). Hdparm (version 3.5). [Computer software]. Retrieved February 27, 2000 from the World Wide Web: http://sunsite.unc.edu/pub/Linux/system/misc.

Maxtor Corporation (1999). Diamond Max 6800 Plus. San Jose, CA: Maxtor Corporation. Retrieved on February 14, 2000 from the World Wide Web: httpd/www.maxtor.com/diamondmaxplus/6800p.html.

Merkey, P. (1999). Beowulf Bulk Data Server. Unpublished manuscript. Greenbelt, MD: Center of Excellence in Space Data and Information Sciences. Retrieved March 10, 2000 from the World Wide Web: http://www.beowulf.org/bds/bds.html

Norcott, W. (1991). IOzone (Version 3.16) [Computer Software]. Nashua, NH.

Oxman, G. (1995). The Extended-2 Filesystem Overview. Unpublished manuscript. Retrieved October 16, 1999 from the World Wide Web: http://metlab.unc.edu/pub/Linux/system/filesystems/Ext2fs-overview-0.1.ps.gz.

Quantum Corporation. (2000). Storage Basics. (2000). Milpitas, CA: 2000 Quantum. Retrieved February 28, 2000 from the World Wide Web: http://www.quantum.com/src/tt/storage_basics.htm.

Risley, D. (n. d.). SCSI. Darien, CT: Internet.com Corp. Unpublished manuscript. Retrieved February 14, 1999 from the World Wide Web: http://www.hardwarecentral.com/hardwarecentral/tutorials/36/3/.

Rubini, A. (1994). Tour of the Linux Kernel Source. Unpublished manuscript. Retrieved October 16, 1999 from the World Wide Web: http://www.infop6.jussieu.fr/Linux/khg/HyperNews/get/tour/tour.html.

Ruemmler, C., & Wilkes, J. (1994). An Introduction to Disk Drive Modeling. Palo Alto, CA: Hewlett-Packard Laboratories.

Rusling, D. A. (1999). The Linux Kernel. Wokingham, Berkshire RG41 3NF, UK. Retrieved October 16, 1999 from the World Wide Web: http://www.wgs.com/LDP/LDP/tlk/.

Seagate Technology, Inc. (1997). Medalist Pro Specifications. (Publication No. 32652-001, Rev. A). Scotts Valley, CA: Seagate Technology, Inc. Retrieved September 15, 1997 from the World Wide Web: http:/www.seagate.com:80/support/disc/manuals/ata/6450pma.pdf.

Seltzer, M., Chen, P., and Ousterhout, J. (1990). Disk Scheduling Revisited. Proceedings 1990 Winter Usenix Conference, Washington, D.C., 313-324.

Vahalia, U. (1996). Unix Internals. Upper Saddle River, NJ: Prentice Hall.

Vongsathorn, P. & Carson, S. (1990). A System for Adaptive Disk Rearrangement. Software Practice and Experience, 20(3), 225-242.

Welsh, M. and Kaufman, L. (1995). Running Linux. Sebastopol, CA: O'Reilly & Associates, Inc.

Western Digital (1999), Detailed Specifications for the WD Enterprise WDE18300, Irvine, CA: Western Digital Corporation. Retrieved March 2, from the World Wide Web: http://www.westerndigital.com/products/drives/specs/wde18300lvds.html

Wirzenius, L. (1995). The Linux System Administrators' Guide. In J. Purcell, & A. Robinson (Eds.), Dr Linux (pp. 323-403). Durham, NC: Red Hat Software, Inc.

APPENDICES

APPENDIX A

Linux System Diagrams

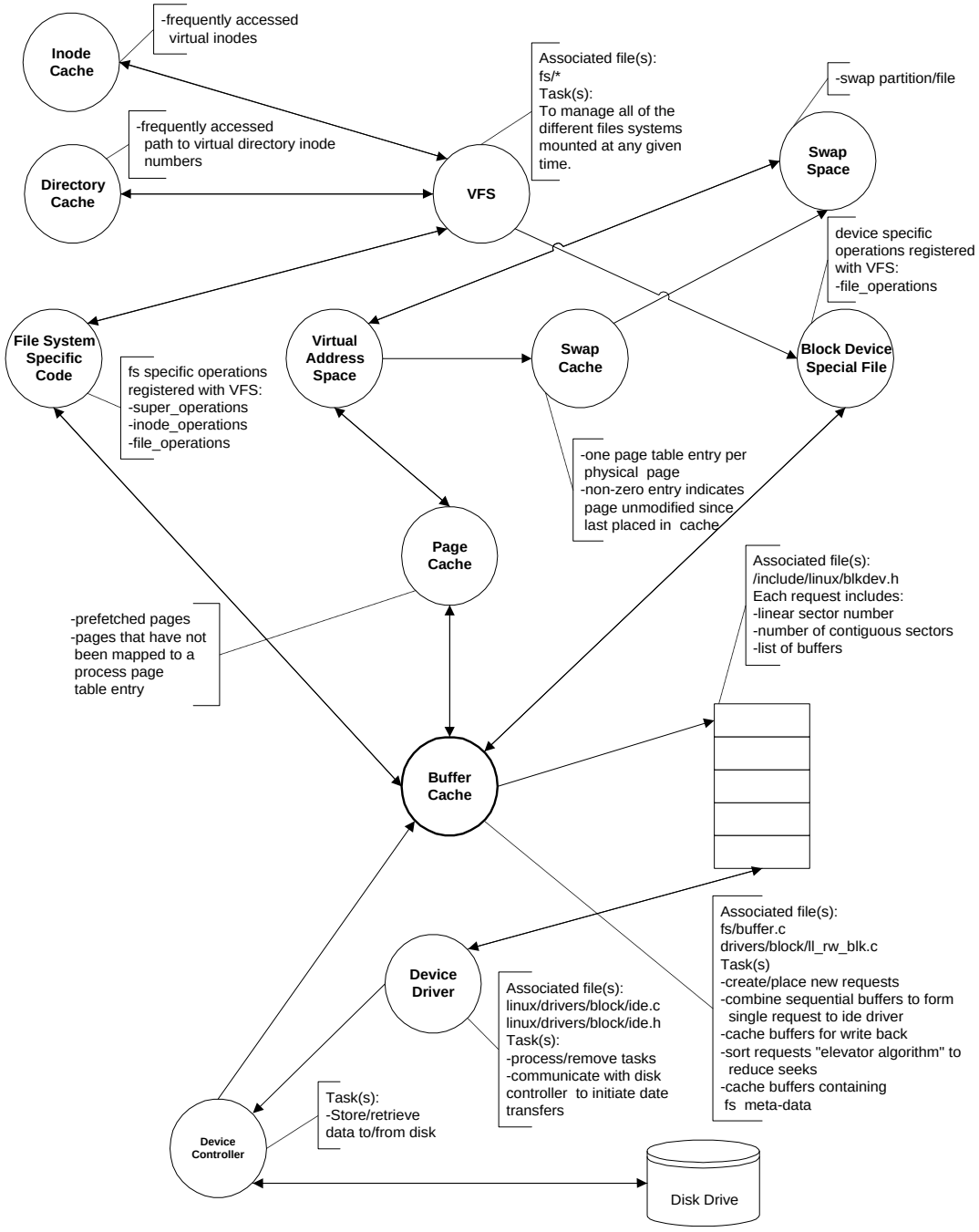


Figure 14. The Linux 2.0 input/output subsystem.

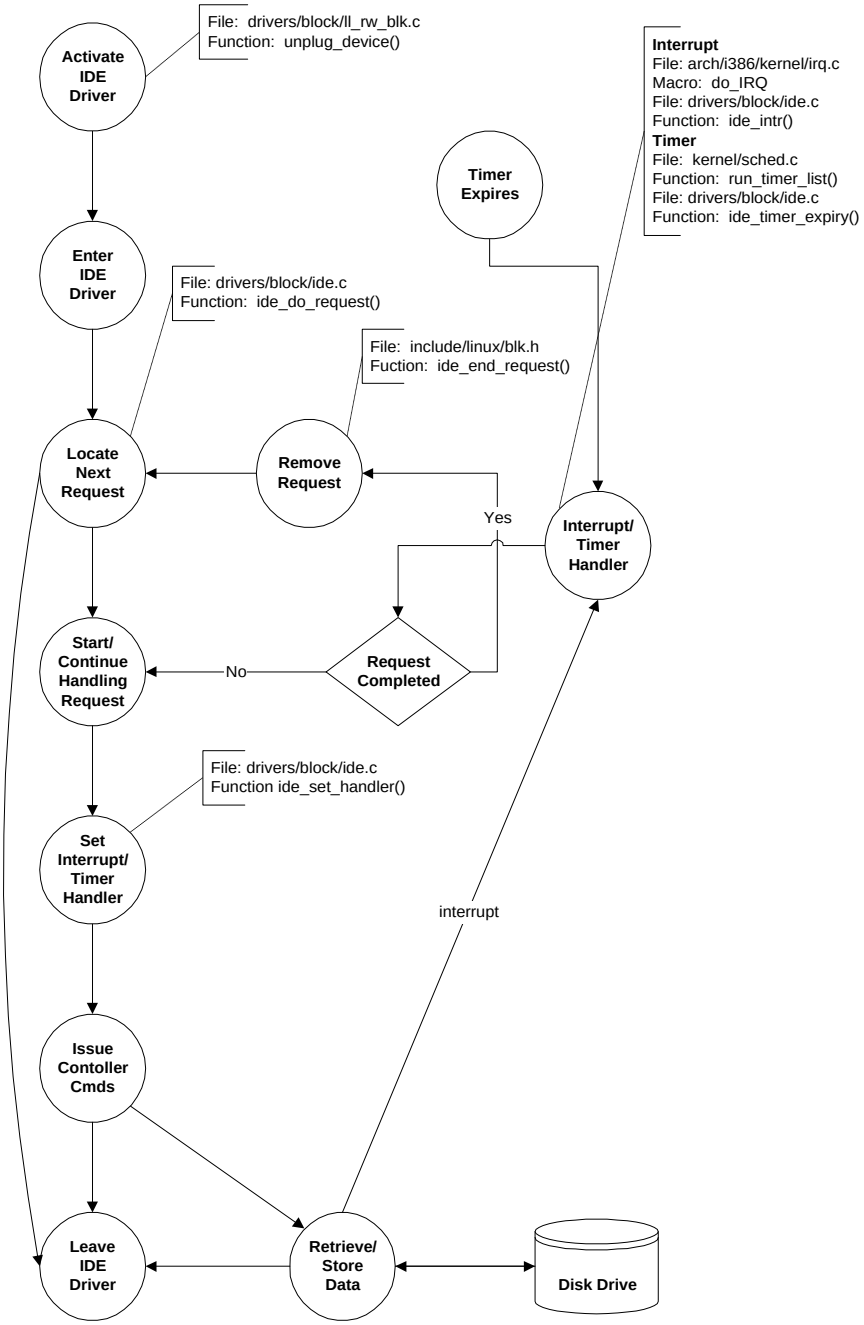


Figure 15. The Linux 2.0 block device driver strategy routine.

APPENDIX B

System Specification Disclosure

CPU:
Model: Intel Celeron
Speed: 450 MHz
Level 1 Cache: 16 KB instruction and 16 KB data caches
Level 2 Cache: 128 KB

Motherboard:
Manufacturer: Micro-Star International Co., Ltd
Model: MS-6131
Chipset: Intel 82440BX AGP
Clock Generator: 66 MHz
BIOS: Award Modular BIOS v 4.51PG W6131F4 V1.1

Main Memory:
Type: 168-pin DIMM
Size: 256 MB DIMM
Speed: 100 MHz

Disk Drive (IDE Configuration Only):
Manufacturer: Maxtor
Model: 92048U8
Capacity: 20.4 GB
Rotational Speed: 72 RPM
Average Seek Time: 9.0 ms
Interface: ATA-4/Ultra DMA
Buffer Cache: 1 MB SDRAM
Approximate Cost: $200

Disk Drive (SCSI Configuration Only):
Manufacturer: Western Digital
Model: WDE18300-004A3
Capacity: 18.4 GB
Rotational Speed: 72 RPM
Average Seek Time: 6.9 ms
Interface: Ultra LVD Wide SCSI
Buffer Cache: 2 MB
Approximate Cost: $600

SCSI Host Adapter (SCSI Configuration Only):
Manufacturer: Adaptec
Model: aha2940UW
Approximate Cost: $300

Figure 16. Hardware specifications associated with the computer system used for performance testing. Sources adapted from Maxtor Corporation (1999). Diamond Max

6800 Plus. San Jose, CA: Maxtor Corporation. Retrieved on February 14, 2000 from the World Wide Web: httpd/www.maxtor.com/diamondmaxplus/6800p.html. Western Digital (1999), Detailed Specifications for the WD Enterprise WDE18300, Irvine, CA: Western Digital Corporation. Retrieved March 2, from the World Wide Web: http://www.westerndigital.com/products/drives/specs/wde18300lvds.html

APPENDIX C

Hard Drive Schematic

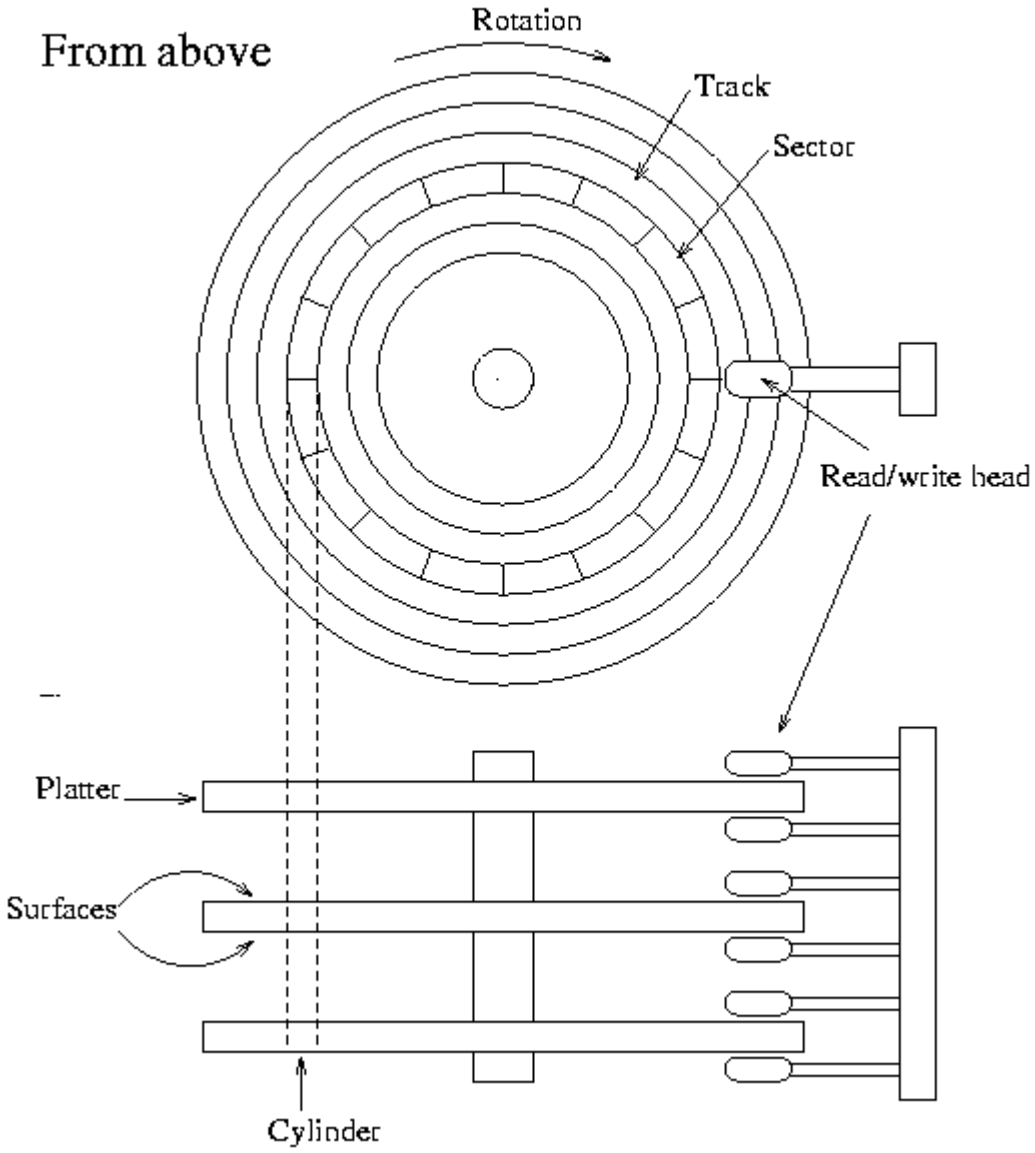


Figure 17. A schematic picture of a hard disk. Source Wirzenius, L. (1995). The Linux System Administrators' Guide. In J. Purcell, & A. Robinson (Eds.), Dr Linux (pp. 323-403). Durham, NC: Red Hat Software, Inc.

VITA

Richard S. Gray

Personal Data:	Date of Birth: February 14, 1963 Place of Birth: Kingsport, Tennessee
Education:	East Tennessee State University, Johnson City, Tennessee; Communications, B.S., 1989 East Tennessee State University, Johnson City, Tennessee; Computer Science, M.S., 2000
Professional Experience:	Graduate Assistant, East Tennessee State University; Johnson City, Tennessee, 1996 – 1997 Senior System Architect, Eolian Inc.; Fairfax, Virginia, 1997 – Present