Performance Effects of Disk Subsystem Choices for VAX† Systems Running 4.2BSD UNIX*

Revised July 27, 1983

Bob Kridle

mt Xinu
2560 9th Street
Suite #312
Berkeley, California 94710

Marshall Kirk McKusick‡

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

Measurements were made of the UNIX file system throughput for various I/O operations using the most attractive currently available Winchester disks and controllers attached to both the native busses (SBI/CMI) and the UNIBUS on both VAX 11/780s and VAX 11/750s. The tests were designed to highlight the performance of single and dual drive subsystems operating in the 4.2BSD *fast file system* environment. Many of the results of the tests were initially counter-intuitive and revealed several important aspects of the VAX implementations which were surprising to us.

The hardware used included two Fujitsu 2351A ‘‘Eagle’’ disk drives on each of two foreign-vendor disk controllers and two DEC RA-81 disk drives on a DEC UDA-50 disk controller. The foreign-vendor controllers were Emulex SC750, SC780 and Systems Industries 9900 native bus interfaced controllers. The DEC UDA-50 controller is a UNIBUS interfaced, heavily buffered controller which is the first implementation of a new DEC storage system architecture, DSA.

One of the most important results of our testing was the correction of several timing parameters in our device handler for devices with an RH750/RH780 type interface and having high burst transfer rates. The correction of these parameters resulted in an increase in performance of over twenty percent in some cases. In addition, one of the controller manufacturers altered their bus arbitration scheme to produce another increase in throughput.

---

†VAX, UNIBUS, and MASSBUS are trademarks of Digital Equipment Corporation.

* UNIX is a trademark of Bell Laboratories.

‡This work was supported under grants from the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

TABLE OF CONTENTS

1. Motivation

2. Equipment
2.1. DEC UDA50 disk controller
2.2. Emulex SC750/SC780 disk controllers
2.3. Systems Industries 9900 disk controller
2.4. DEC RA81 disk drives
2.5. Fujitsu 2351A disk drives

3. Methodology

4. Tests

5. Results

6. Conclusions

Acknowledgements

References

Appendix A
A.1. read_8192
A.2. write_4096
A.3. write_8192
A.4. rewrite_8192

1. Motivation

These benchmarks were performed for several reasons. Foremost was our desire to obtain guideline to aid in choosing one the most expensive components of any VAX UNIX configuration, the disk storage system. The range of choices in this area has increased dramatically in the last year. DEC has become, with the introduction of the UDA50/RA81 system, cost competitive in the area of disk storage for the first time. Emulex's entry into the VAX 11/780 SBI controller field, the SC780, represented a important choice for us to examine, given our previous success with their VAX 11/750 SC750 controller and their UNIBUS controllers. The Fujitsu 2351A Winchester disk drive represents the lowest cost-per-byte disk storage known to us. In addition, Fujitsu's reputation for reliability was appealing. The many attractive aspects of these components justified a more careful examination of their performance aspects under UNIX.

In addition to the direct motivation of developing an effective choice of storage systems, we hoped to gain more insight into VAX UNIX file system and I/O performance in general. What generic characteristics of I/O subsystems are most important? How important is the location of the controller on the SBI/CMI versus the UNIBUS? Is extensive buffering in the controller essential or even important? How much can be gained by putting more of the storage system management and optimization function in the controller as DEC does with the UDA50?

We also wanted to resolve particular speculation about the value of storage system optimization by a controller in a UNIX environment. Is the access optimization as effective as that already provided by the existing 4.2BSD UNIX device handlers for traditional disks? VMS disk handlers do no seek optimization. This gives the UDA50 controller an advantage over other controllers under VMS which is not likely to be as important to UNIX. Are there penalties associated with greater intelligence in the controller?

A third and last reason for evaluating this equipment is comparable to the proverbial mountain climbers answer when asked why he climbs a particular mountain, ‘‘It was there.’’ In our case the equipment was there. We were lucky enough to assemble all the desired disks and controllers and get them installed on a temporarily idle VAX 11/780. This got us started collecting data. Although many of the tests were later rerun on a variety of other systems, this initial test bed was essential for working out the testing bugs and getting our feet wet.

2. Equipment

Various combinations of the three manufacturers disk controllers, and two pairs of Winchester disk drives were tested on both VAX 11/780 and VAX 11/750 CPUs. The Emulex and Systems Industries disk controllers were interfaced to Fujitsu 2351A "Eagle" 404 Megabyte disk drives. The DEC UDA50 disk controller was interfaced to two DEC RA81 456 Megabyte Winchester disk drives. All three controllers were tested on the VAX 780 although only the Emulex and DEC controllers were benchmarked on the VAX 11/750. Systems Industries makes a VAX 11/750 CMI interface for their controller, but we did not have time to test this device. In addition, not all the storage systems were tested for two drive throughput. Each of the controllers and disk drives used in the benchmarks is described briefly below.

2.1. DEC UDA50 disk controller

This is a new controller design which is part of a larger, long range storage architecture referred to as "DSA" or **D**igital **S**torage **A**rchetecture. An important aspect of DSA is migrating a large part of the storage management previously handled in the operating system to the storage system. Thus, the UDA50 is a much more intelligent controller than previous interfaces like the RH750 or RH780. The UDA50 handles all error correction. It also deals with most of the physical storage parameters. Typically, system software requests a logical block or sequence of blocks. The physical locations of these blocks, their head, track, and cylinder indices, are determined by the controller. The UDA50 also orders disk requests to maximize throughput where possible, minimizing total seek and rotational delays. Where multiple drives are attached to a single controller, the UDA50 can interleave simultaneous data transfers from multiple drives.

The UDA50 is a UNIBUS implementation of a DSA controller. It contains 52 sectors of internal buffering to minimize the effects of a slow UNIBUS such as the one on the VAX-11/780. This buffering also minimizes the effects of contention with other UNIBUS peripherals.

2.2. Emulex SC750/SC780 disk controllers

These two models of the same controller interface to the CMI bus of a VAX 11/750 and the SBI bus of a 11/VAX 780, respectively. To the operating system, they emulate either an RH750 or and RH780. The controllers install in the MASSBUS locations in the CPU cabinets and operate from the VAX power suplies. They provide an "SMD" or **S**torage **M**odule **D**rive interface to the disk drives. Although a large number of disk drives use this interface, we tested the controller exclusively connected to Fujitsu 2351A disks.

The controller ws first implemented for the VAX-11/750 as the SC750 model several years ago. Although the SC780 was introduced more recently, both are stable products with no bugs known to us.

2.3. System Industries 9900 disk controller

This controller is an evolution of the S.I. 9400 first introduced as a UNIBUS SMD interface. The 9900 has been enhanced to include an interface to the VAX 11/780 native bus, the SBI. It has also been upgraded to operate with higher data rate drives such as the Fujitsu 2351As we used in this test. The controller is contained in its own rack-mounted drawer with an integral power supply. The interface to the SMD is a four module set which mounts in a CPU cabinet slot normally occupied by an RH780. The SBI interface derives power from the VAX CPU cabinet power supplies.

2.4. DEC RA81 disk drives

The RA81 is a rack-mountable 456 Megabyte (formatted) Winchester disk drive manufactured by DEC. It includes a great deal of technology which is an integral part of the DEC **DSA** scheme. The novel technology includes a serial packet based communications protocol with the controller over a pair of mini-coaxial cables. The physical characteristics of the RA81 are shown in the table below:

DEC RA81 Disk Drive Characteristics	
Peak Transfer Rate	2.2 Mbytes/sec.
Rotational Speed	3,600 RPM
Data Sectors/Track	51
Logical Cylinders	1,248
Logical Data Heads	14
Data Capacity	456 Mbytes
Minimum Seek Time	6 milliseconds
Average Seek Time	28 milliseconds
Maximum Seek Time	52 milliseconds

2.5. Fujitsu 2351A disk drives

The Fujitsu 2351A disk drive is a Winchester disk drive with an SMD controller interface. Fujitsu has developed a very good reputation for reliable storage products over the last several years. The 2351A has the following physical characteristics:

Fujitsu 2351A Disk Drive Characteristics	
Peak Transfer Rate	1.859 Mbytes/sec.
Rotational Speed	3,961 RPM
Data Sectors/Track	48
Cylinders	842
Data Heads	20
Data Capacity	404 Mbytes
Minimum Seek Time	5 milliseconds
Average Seek Time	18 milliseconds
Maximum Seek Time	35 milliseconds

3. Methodology

Our goal was to evaluate the performance of the target peripherals in an environment as much like our 4.2BSD UNIX systems as possible. There are two basic approaches to creating this kind of test environment. These might be termed the *indirect* and the *direct* approach. The approach used by DEC in producing most of the performance data on the UDA50/RA81 system under VMS is what we term the indirect approach. We chose to use the direct approach.

The indirect approach used by DEC involves two steps. First, the environment in which performance is to be evaluated is parameterized. In this case, the disk I/O characteristics of VMS were measured as to the distribution of various sizes of accesses and the proportion of reads and writes. This parameterization of typical I/O activity was termed a ‘‘vax mix.’’ The second stage involves simulating this mixture of I/O activities with the devices to be tested and noting the total volume of transactions processed per unit time by each system.

The problems encountered with this indirect approach often have to do with the completeness and correctness of the parameterization of the context environment. For example, the ‘‘vax mix’’ model constructed for DECs tests uses a random distribution of seeks to the blocks read or written. It is not likely that any real system produces a distribution of disk transfer locations which is truly random and does not exhibit strong locality characteristics.

The methodology chosen by us is direct in the sense that it uses the standard structured file system mechanism present in the 4.2BSD UNIX operating system to create the sequence of locations and sizes of reads and writes to the benchmarked equipment. We simply create, write, and read files as they would be by user’s activities. The disk space allocation and disk cacheing mechanism built into UNIX is used to produce the actual device reads and writes as well as to determine their size and location on the disk. We measure and compare the rate at which these *user files* can be written, rewritten, or read.

The advantage of this approach is the implicit accuracy in testing in the same environment in which the peripheral will be used. Although this system does not account for the I/O produced by some paging and swapping, in our memory rich environment these activities account for a relatively small portion of the total disk activity.

A more significant disadvantage to the direct approach is the occasional difficulty we have in accounting for our measured results. The apparently straight-forward activity of reading or writing a logical file on disk can produce a complex mixture of disk traffic. File I/O is supported by a file management system that buffers disk traffic through an internal cache, which allows writes to ba handled asynchronously. Reads must be done synchronously, however this restriction is moderated by the use of read-ahead. Small changes in the performance of the disk controller subsystem can result in large and unexpected changes in the file system performance, as it may change the characteristics of the memory contention experienced by the processor.

4. Tests

Our battery of tests consists of four programs, read_8192, write_8192, write_4096 and rewrite_8192 originally written by [McKusick83] to evaluate the performance of the new file system in 4.2BSD. These programs all follow the same model and are typified by read_8192 shown here.

```
#define     BUFSIZ 8192
main( argc, argv)
char **argv;
{
        char buf[BUFSIZ];
        int i, j;

        j = open(argv[1], 0);
        for (i = 0; i < 1024; i++)
                read(j, buf, BUFSIZ);
}
```

The remaining programs are included in appendix A.

These programs read, write with two different blocking factors, and rewrite logical files in structured file system on the disk under test. The write programs create new files while the rewrite program overwrites an existing file. Each of these programs represents an important segment of the typical UNIX file system activity with the read program representing by far the largest class and the rewrite the smallest.

A blocking factor of 8192 is used by all programs except write_4096. This is typical of most 4.2BSD user programs since a standard set of I/O support routines is commonly used and these routines buffer data in similar block sizes.

For each test run, a empty eight Kilobyte block file system was created in the target storage system. Then each of the four tests was run and timed. Each test was run three times; the first to clear out any useful data in the cache, and the second two to insure that the experiment had stablized and was repeatable. Each test operated on eight Megabytes of data to insure that the cache did not overly influence the results. Another file system was then initialized using a basic blocking factor of four Kilobytes and the same tests were run again and timed. A command script for a run appears as follows:

```
#!/bin/csh
set time=2
echo "8K/1K file system"
newfs /dev/rhp0g eagle
mount /dev/hp0g /mnt0
mkdir /mnt0/foo
echo "write_8192 /mnt0/foo/tst2"
rm -f /mnt0/foo/tst2
write_8192 /mnt0/foo/tst2
rm -f /mnt0/foo/tst2
write_8192 /mnt0/foo/tst2
rm -f /mnt0/foo/tst2
write_8192 /mnt0/foo/tst2
echo "read_8192 /mnt0/foo/tst2"
read_8192 /mnt0/foo/tst2
read_8192 /mnt0/foo/tst2
read_8192 /mnt0/foo/tst2
umount /dev/hp0g
```

5. Results

The following tables indicate the results of our test runs. Note that each table contains results for tests run on two varieties of 4.2BSD file systems. The first set of results is always for a file system with a basic blocking factor of eight Kilobytes and a fragment size of 1 Kilobyte. The second sets of measurements are for file systems with a four Kilobyte block size and a one Kilobyte fragment size. The values in parenthesis indicate the percentage of CPU time used by the test program. In the case of the two disk arm tests, the value in parenthesis indicates the sum of the percentage of the test programs that were run. Entries of ‘‘n. m.’’ indicate this value was not measured.

4.2BSD File Systems Tests - **VAX 11/750**				
Logically Sequential Transfers from an **8K/1K** 4.2BSD File System (Kbytes/sec.)				
Test	Emulex SC750/Eagle		UDA50/RA81	
	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	490 (69%)	620 (96%)	310 (44%)	520 (65%)
write_4096	380 (99%)	370 (99%)	370 (97%)	360 (98%)
write_8192	470 (99%)	470 (99%)	320 (71%)	410 (83%)
rewrite_8192	650 (99%)	620 (99%)	310 (50%)	450 (70%)
Logically Sequential Transfers from **4K/1K** 4.2BSD File System (Kbytes/sec.)				
Test	Emulex SC750/Eagle		UDA50/RA81	
	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	300 (60%)	400 (84%)	210 (42%)	340 (77%)
write_4096	320 (98%)	320 (98%)	220 (67%)	290 (99%)
write_8192	340 (98%)	340 (99%)	220 (65%)	310 (98%)
rewrite_8192	450 (99%)	450 (98%)	230 (47%)	340 (78%)

Note that the rate of write operations on the VAX 11/750 are ultimately CPU limited in some cases. The write rates saturate the CPU at a lower bandwidth than the reads because they must do disk allocation in addition to moving the data from the user program to the disk. The UDA50/RA81 saturates the CPU at a lower transfer rate for a given operation than the SC750/Eagle because it causes more memory contention with the CPU. We do not know if this contention is caused by the UNIBUS controller or the UDA50.

The following table reports the results of test runs on a VAX 11/780 with 4 Megabytes of main memory.

4.2BSD File Systems Tests - **VAX 11/780**						
Logically Sequential Transfers from an **8K/1K** 4.2BSD File System (Kbytes/sec.)						
Test	Emulex SC780/Eagle		UDA50/RA81		Sys. Ind. 9900/Eagle	
	1 Drive	2 Drives	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	560 (70%)	480 (58%)	360 (45%)	540 (72%)	340 (41%)	520 (66%)
write_4096	440 (98%)	440 (98%)	380 (99%)	480 (96%)	490 (96%)	440 (84%)
write_8192	490 (98%)	490 (98%)	220 (58%)*	480 (92%)	490 (80%)	430 (72%)
rewrite_8192	760 (100%)	560 (72%)	220 (50%)*	180 (52%)*	490 (60%)	520 (62%)
Logically Sequential Transfers from an **4K/1K** 4.2BSD File System (Kbytes/sec.)						
Test	Emulex SC780/Eagle		UDA50/RA81		Sys. Ind. 9900/Eagle	
	1 Drive	2 Drives	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	490 (77%)	370 (66%)	n.m.	n.m.	200 (31%)	370 (56%)
write_4096	380 (98%)	370 (98%)	n.m.	n.m.	200 (46%)	370 (88%)
write_8192	380 (99%)	370 (97%)	n.m.	n.m.	200 (45%)	320 (76%)
rewrite_8192	490 (87%)	350 (66%)	n.m.	n.m.	200 (31%)	300 (46%)

* the operation of the hardware was suspect during these tests.

The dropoff in reading and writing rates for the two drive SC780/Eagle tests are probably due to the file system using insufficient rotational delay for these tests. We have not fully investigated these times.

The following table compares data rates on VAX 11/750s directly with those of VAX 11/780s using the UDA50/RA81 storage system.

4.2BSD File Systems Tests - **DEC UDA50 - 750 vs. 780**				
Logically Sequential Transfers from an **8K/1K** 4.2BSD File System (Kbytes/sec.)				
Test	VAX 11/750 UNIBUS		VAX 11/780 UNIBUS	
	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	310 (44%)	520 (84%)	360 (45%)	540 (72%)
write_4096	370 (97%)	360 (100%)	380 (99%)	480 (96%)
write_8192	320 (71%)	410 (96%)	220 (58%)*	480 (92%)
rewrite_8192	310 (50%)	450 (80%)	220 (50%)*	180 (52%)*
Logically Sequential Transfers from an **4K/1K** 4.2BSD File System (Kbytes/sec.)				
Test	VAX 11/750 UNIBUS		VAX 11/780 UNIBUS	
	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	210 (42%)	342 (77%)	n.m.	n.m.
write_4096	215 (67%)	294 (99%)	n.m.	n.m.
write_8192	215 (65%)	305 (98%)	n.m.	n.m.
rewrite_8192	227 (47%)	336 (78%)	n.m.	n.m.

* the operation of the hardware was suspect during these tests.

The higher throughput available on VAX 11/780s is due to a number of factors. The larger main memory size allows a larger file system cache. The block allocation routines run faster, raising the upper limit on the data rates in writing new files.

The next table makes the same comparison using an Emulex controller on both systems.

4.2BSD File Systems Tests - **Emulex - 750 vs. 780**				
Logically Sequential Transfers from an **8K/1K** 4.2BSD File System (Kbytes/sec.)				
Test	VAX 11/750 CMI Bus		VAX 11/780 SBI Bus	
	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	490 (69%)	620 (96%)	560 (70%)	480 (58%)
write_4096	380 (99%)	370 (99%)	440 (98%)	440 (98%)
write_8192	470 (99%)	470 (99%)	490 (98%)	490 (98%)
rewrite_8192	650 (99%)	620 (99%)	760 (100%)	560 (72%)
Logically Sequential Transfers from an **4K/1K** 4.2BSD File System (Kbytes/sec.)				
Test	VAX 11/750 CMI Bus		VAX 11/780 SBI Bus	
	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	300 (60%)	400 (84%)	490 (77%)	370 (66%)
write_4096	320 (98%)	320 (98%)	380 (98%)	370 (98%)
write_8192	340 (98%)	340 (99%)	380 (99%)	370 (97%)
rewrite_8192	450 (99%)	450 (98%)	490 (87%)	350 (66%)

The following table illustrates the evolution of our testing process as both hardware and software problems effecting the performance of the Emulex SC780 were corrected. The software change was suggested to us by George Goble of Purdue University.

The 4.2BSD handler for RH750/RH780 interfaced disk drives contains several constants which to determine how much time is provided between an interrupt signaling the completion of a positioning command and the subsequent start of a data transfer operation. These lead times are expressed as sectors of rotational delay. If they are too small, an extra complete rotation will often be required between a seek and subsequent read or write operation. The higher bit rate and rotational speed of the 2351A Fujitsu disk drives required increasing these constants.

The hardware change involved allowing for slightly longer delays in arbitrating for cycles on the SBI bus by starting the bus arbitration cycle a little further ahead of when the data was ready for transfer. Finally we had to increase the rotational delay between consecutive blocks in the file because the higher bandwidth from the disk generated more memory contention, which slowed down the processor.

4.2BSD File Systems Tests - **Emulex SC780 Disk Controller Evolution**						
Logically Sequential Transfers from an **8K/1K** 4.2BSD File System (Kbytes/sec.)						
Test	Inadequate Search Lead Initial SBI Arbitration		OK Search Lead Init SBI Arb.		OK Search Lead Improved SBI Arb.	
	1 Drive	2 Drives	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	320	370	440 (60%)	n.m.	560 (70%)	480 (58%)
write_4096	250	270	300 (63%)	n.m.	440 (98%)	440 (98%)
write_8192	250	280	340 (60%)	n.m.	490 (98%)	490 (98%)
rewrite_8192	250	290	380 (48%)	n.m.	760 (100%)	560 (72%)
Logically Sequential Transfers from an **4K/1K** 4.2BSD File System (Kbytes/sec.)						
Test	Inadequate Search Lead Initial SBI Arbitration		OK Search Lead Init SBI Arb.		OK Search Lead Improved SBI Arb.	
	1 Drive	2 Drives	1 Drive	2 Drives	1 Drive	2 Drives
read_8192	200	220	280	n.m.	490 (77%)	370 (66%)
write_4096	180	190	300	n.m.	380 (98%)	370 (98%)
write_8192	180	200	320	n.m.	380 (99%)	370 (97%)
rewrite_8192	190	200	340	n.m.	490 (87%)	350 (66%)

6. Conclusions

Peak available throughput is only one criterion in most storage system purchasing decisions. Most of the VAX UNIX systems we are familiar with are not I/O bandwidth constrained. Nevertheless, an adequate disk bandwidth is necessary for good performance and especially to preserve snappy response time. All of the disk systems we tested provide more than adequate bandwidth for typical VAX UNIX system application. Perhaps in some I/O-intensive applications such as image processing, more consideration should be given to the peak throughput available. In most situations, we feel that other factors are more important in making a storage choice between the systems we tested. Cost, reliability, availability, and support are some of these factors. The maturity of the technology purchased must also be weighed against the future value and expandability of newer technologies.

Two important conclusions about storage systems in general can be drawn from these tests. The first is that buffering can be effective in smoothing the effects of lower bus speeds and bus contention. Even though the UDA50 is located on the relatively slow UNIBUS, its performance is similar to controllers located on the faster processor busses. However, the SC780 with only one sector of buffering shows that little buffering is needed if the underlying bus is fast enough.

Placing more intelligence in the controller seems to hinder UNIX system performance more than it helps. Our profiling tests have indicated that UNIX spends about the same percentage of time in the SC780 driver and the UDA50 driver (about 10-14%). Normally UNIX uses a disk sort algorithm that separates reads and writes into two seek order queues. The read queue has priority over the write queue, since reads cause processes to block, while writes can be done asynchronously. This is particularly useful when generating large files, as it allows the disk allocator to read new disk maps and begin doing new allocations while the blocks allocated out of the previous map are written to disk. Because the UDA50 handles all block ordering, and because it keeps all requests in a single queue, there is no way to force the longer seek needed to get the next disk map. This disfunction causes all the writes to be done before the disk map read, which idles the disk until a new set of blocks can be allocated.

The additional functionality of the UDA50 controller that allows it to transfer simultaneously from two drives at once tends to make the two drive transfer tests run much more effectively. Tuning for the single drive case works more effectively in the two drive case than when controllers that cannot handle simultaneous transfers are used.

7. Acknowledgements

We thank Paul Massigilia and Bill Grace of Digital Equipment Corp for helping us run our disk tests on their UDA50/RA81. We also thank Rich Notari and Paul Ritkowski of Emulex for making their machines available to us to run our tests of the SC780/Eagles. Dan McKinster, then of Systems Industries, arranged to make their equipment available for the tests. We appreciate the time provided by Bob Gross, Joe Wolf, and Sam Leffler on their machines to refine our benchmarks. Finally we thank our sponsors, the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

8. References

[McKusick83] M. McKusick, W. Joy, S. Leffler, R. Fabry, ‘‘A Fast File System for UNIX’’, *ACM Transactions on Computer Systems 2*, 3. pp 181-197, August 1984.

9. Appendix A

9.1. read_8192

```
#define	BUFSIZ 8192
main( argc, argv)
char **argv;
{
	char buf[BUFSIZ];
	int i, j;

	j = open(argv[1], 0);
	for (i = 0; i < 1024; i++)
		read(j, buf, BUFSIZ);
}
```

9.2. write_4096

```
#define	BUFSIZ 4096
main( argc, argv)
char **argv;
{
	char buf[BUFSIZ];
	int i, j;

	j = creat(argv[1], 0666);
	for (i = 0; i < 2048; i++)
		write(j, buf, BUFSIZ);
}
```

9.3. write_8192

```
#define	BUFSIZ 8192
main( argc, argv)
char **argv;
{
	char buf[BUFSIZ];
	int i, j;

	j = creat(argv[1], 0666);
	for (i = 0; i < 1024; i++)
		write(j, buf, BUFSIZ);
}
```

9.4. rewrite_8192

```
#define	BUFSIZ 8192
main( argc, argv)
char **argv;
{
	char buf[BUFSIZ];
	int i, j;

	j = open(argv[1], 2);
	for (i = 0; i < 1024; i++)
		write(j, buf, BUFSIZ);
}
```