COEN 177: Operating Systems
Lab assignment 9: File Performance Measurement

Objectives

1. To evaluate file system performance in the face of sequential I/O requests, mainly disk reads and writes.
2. To evaluate the impact of multiple competing threads attempting to read/write simultaneously.

Guidelines

The goal of this assignment is to gain experience with evaluation of the performance of the file system. Specifically, you will be timing various I/O intensive processes. I/O performance can be affected by more than the volume of data being moved. For example, it can be affected by the size of the individual requests being made, whether the requests are reading or writing data, and by the degree of contention for access to the disk. It can also be affected dramatically by the pattern of data access (e.g., whether it is sequential or random)--although we will only be looking at sequential access in this assignment.

Test files

Step 1. Create files of random data of sizes 100KB, 1MB, 10MB, and 100MB. You may use “cat” and “head” commands (e.g. **cat /dev/urandom | head -c 100000**). /dev/urandom and /dev/random are special files that serve as pseudorandom number generators, with /dev/random being much slower but cryptographically secure. “cat” is used to output the content, which will be unprintable random binary, and “head” is used to display the specified number of lines, or with the -c option, the number of bytes. Here, the result of “cat” is sent to the upstream end of the pipe and “head” receives these results and truncates them at the given byte count. You should redirect the command to a new file using the > symbol. Be aware that as a student you are likely to have a disk quota of 20GB for all of your files, **including preexisting files**, and if you reach that quota it may be very difficult to return below it (as your applications, including GUI file explorers, also make use of that quota to run, making it difficult to delete the over-quota files). As long as you delete any copied output files regularly and do not create files in excess of 100MB, you should be fine.
You may write the following commands in .sh file.

```
#!/bin/bash
cat /dev/urandom | head -c 100000 > file1.bin
cat /dev/urandom | head -c 1000000 > file2.bin
cat /dev/urandom | head -c 10000000 > file3.bin
cat /dev/urandom | head -c 100000000 > file4.bin
```

Check the size of the files with the command **ls -l file?.bin**

C programs

Step 2. Write a C program to read the files your created in Step 1 from beginning to end, then measure how long does your program take to read each file. You may pass the name of the file as a command line argument “argv[1]”. Use a buffer of size 10000 bytes for each read operation. You may use the following code snippet:

```
char buffer[10000];
FILE *fp;
fp = fopen(argv[1], "rb");
while (fread(buffer, sizeof(buffer), 1, fp)){
}
fclose(fp);
```

The “time” command can be used to determine how long a given command/ program takes to run. It returns three values:

- real: total time from the moment you hit the Enter key until the moment the command is completed
- user: CPU time spent in user mode
- sys: CPU time spent in kernel mode

Use real value to measure the I/O performance. This can be implemented by reading the return real value of the command:

```
time ./Step2 file1.bin
```

To measure time for all files, you may write the following commands in a .sh file

```
for file in file1.bin file2.bin file3.bin file4.bin
do
   echo "Step2 $file"
   time ./Step2 $file
   echo " "
done
```

Step 3. Make a copy of your program from Step 2 and modify it to measure the I/O performance for 100, 1,000, 10,000 and 100,000 bytes of buffer sizes. Read the buffer sizes from the command line as "argv[2]".

To measure the time for all files and for different buffer sizes, use the following commands in a .sh file, **Note that the rm command does not do anything for step 3, it will be relevant in step 4.**

```
for file in file1.bin file2.bin file3.bin file4.bin
do
        for buffer in 100 1000 10000 100000
        do
        echo "Step3 $file $buffer"
        time ./Step3 $file $buffer
        rm -f copy.out
        echo " "
        done
done
```

Step 4. Make a copy of your program in Step 3 and modify it so that a write operation is made to a newly created file for each read operation. In other words, you will now be measuring the I/O performance by timing your program that copies each file to a new file. So now you are testing the speed of sequential reads+writes for files of varying size, and using I/O operations of varying size.

Copy the .sh file from Step 3 with modifications to reference the new program.

Step 5. Make a copy of your program in Step 4 and modify it so that multiple copies of each file are made. This should be implemented in your program by creating multiple concurrent threads, each of which **independently** reads and copies the input file. You may set the number of threads based on a value from the command line "argv[3]".

Measure the I/O performance for 2, 8, 32, and 64 concurrent threads where each will read/write files of different sizes using buffers of different sizes as well. please note in your final submission and submit the results of what you were able to test.

Write a .sh file as follows (for the rm command, the * character is a wildcard that matches any string of characters including the empty string, the ? character is a wildcard that matches any one character)

```
for file in file1.bin file2.bin file3.bin file4.bin
do
        for buffer in 100 1000 10000 100000
        do
                for thread in 2 8 32 64
                do
                        echo "Step5 $file $buffer $thread"
                        time ./Step5 $file $buffer $thread
                        rm copy*.out
                        echo " "
                done
        done
done
```

Step 6. Use the “make” command to compile all your c programs (Steps 2 – 5) in one command. To do so, create a Makefile and include the following statements:

```
all: Step2.c Step3.c Step4.c Step5.c
    gcc -o Step2 Step2.c
    gcc -o Step3 Step3.c
    gcc -o Step4 Step4.c
    gcc -o Step5 Step5.c -lpthread

clean:; rm -f *.out Step2 Step3 Step4 Step5
```

Requirements to complete the lab

1. Write and discuss the measurements of I/O performance in Steps 2 – 5 using all the buffer sizes specified, covering general trends **not** the exact numbers, except as limited examples, and upload your report on Camino
2. Demo to TA the following steps

a. ls -l file?.bin and show all your binary files.
b. Run **make** to compile all your code
c. Run **time** on each program on one configuration each (selected by the TA) for steps 4 and 5, using **ls -l file?.bin copy*.out** and **diff --from-file=file1.bin copy*.out** (replace file1.bin with the appropriate input file) for each. The **diff** command should produce no output if there are no differences (for details, run **man diff**). **Remove all copies after running each diff command.**
d. Run each shell script for steps 2-5.
e. Run **make clean** to remove compiled files.