**CS-GY 6233 Final Project**

**Binvant Bajwa (bsb9397) and Chinmay Kulkarni (cmk9709)**

**Contents**

1. **How to run the project**
2. **Measurement**
3. **Raw Performance**
4. **Caching**
5. **System calls**
6. **Raw Performance**
7. **Time run**
8. **How to run the project and get all desired results:**

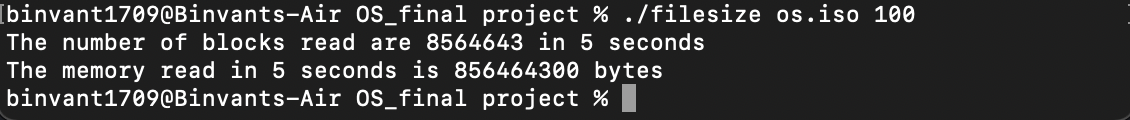
Run the commands given below to test out all the functionalities of our project.

1. ./build
2. ./run <filename> <r|w> <block\_size><block\_count>
3. ./fast <filename>
4. ./filesize <filename><block\_count>
5. ./timerun <filename><How much time you want to run the file in ms>

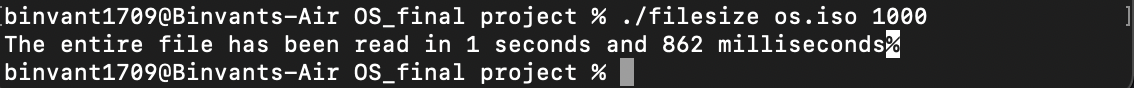
P.S: While writing to a file, if you face a permission error while opening the file please run the command **chmod 600 <name of file to which you have written>**. That will solve the error.

1. **Measurement**

Here, we tried our code on the ubuntu image file given by the professor. The goal here was to find a file size which took a sufficiently long time to read when given a block size. Hence, we tried to measure how much of the file is read in the first 5 seconds when input given is the file name & block size. For our experiment, performed on our local MacBook air M1 on the ubuntu image file, a total of 8564643 blocks are read in 5 seconds as per the screenshot.



But, if our block size is large enough to read the whole file, we get the following output:



**Extra credit idea:**

We made a small 70 mb file to test our program against the dd command in Linux. These were the results:

Text

Description automatically generated

When the block size was 100, our program performed almost like dd.

1. **Raw performance**

Here, we output the performance in Megabytes/second (MiB/s) and make a performance graph which shows how our performance changes with different block sizes.

Table

Description automatically generated

1. **Caching**

In the below screenshot, we can see the effect of caching. Here, the run time decreases by a very small margin and performance increases when we keep running the command again and again.

Text

Description automatically generated

After we clear the cache using sudo purge, we get the following output:

Text

Description automatically generated

Here,

The performance decreases to 337.709 MiB/s after we clear the cache and run it again.

Chart, line chart

Description automatically generated

The above graph shows the Cached Vs Non cached performance of our program. We varied the block size from 50 to 50000 and in all the cases we can clearly see that cached performance is slightly better than the non-cached performance.

**Extra credit idea:** Why 3 in sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop\_caches"?

We can use various forms of this command to either free dentries, inodes or page cache.

When we run sudo sh -c "/usr/bin/echo 1 > /proc/sys/vm/drop\_cache, Page cache is freed. The page cache could contain any memory mappings to blocks on disk. That could conceivably be buffered I/O, memory mapped files, paged areas of executables--anything that the OS could hold in memory from a file.

When we run sudo sh -c "/usr/bin/echo 2 > /proc/sys/vm/drop\_cache, dentries and inodes are freed. An inode is a data structure that represents a file. A dentries is a data structure that represents a directory. These structures could be used to build a memory cache that represents the file structure on a disk.

And when we run sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop\_cache, page cache, dentries and inodes all are freed altogether.

1. **System calls**

Here, when the block size is 1 byte, the speed and time taken to read and XOR the whole file is as shown below:

Graphical user interface, diagram, text

Description automatically generated

For the whole file-

Total blocks read: 704684545

Total time in seconds: 231.28

So, total system calls = blocks read/time = 3046889.246 system calls/s.

1. **Raw performance**

Here, we found that the most optimal block size for our experiment was 30,000. If we go on increasing the block size after that we will see that there is not much improvement in performance and the saturation point is reached.

For block size = 30000:

Text

Description automatically generated

For block size= 40000:

Text

Description automatically generated

So, by increasing our block size here by 10,000, the time difference is negligible at this point so we can say that 30000 is a good estimate for optimal block size.

1. **Time run**

We implemented an extra functionality where you can give time as an input and determine how much of your file was read in that time. Time must be input in milliseconds.

Text

Description automatically generated