CS-GY 6233 Final Project

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

Contents

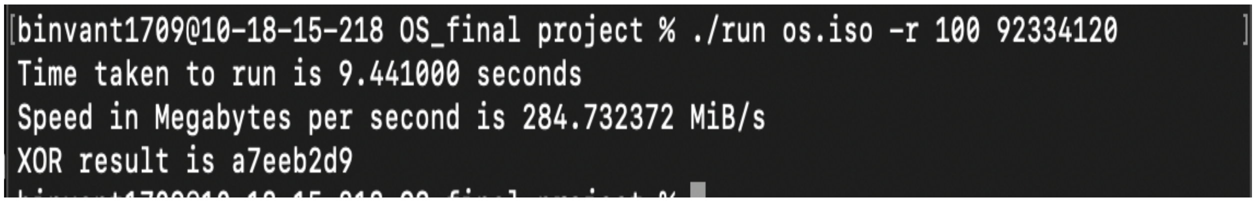
1.)How to run the project and get all desired results:

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.) ./run2 <filename><block_size>
4.) ./fast <filename>
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.
We refer MiB/s as Megabytes/second throughout this report.

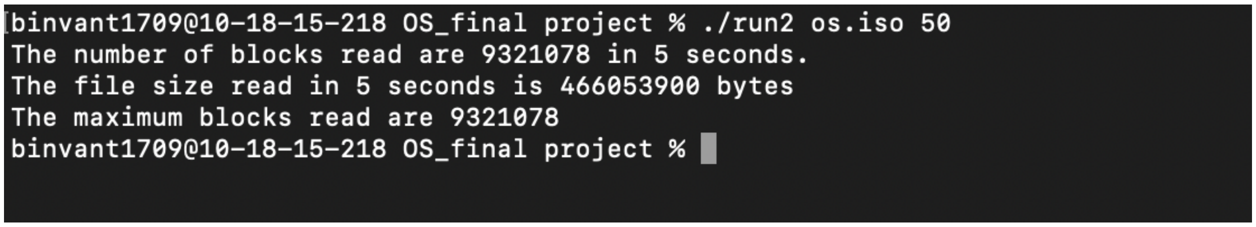
When we run the ./run command, this is the output we get:



```
binvant1709@10-18-15-218 OS_final project % ./run os.iso -r 100 92334120
Time taken to run is 9.441000 seconds
Speed in Megabytes per second is 284.732372 MiB/s
XOR result is a7eeb2d9
```

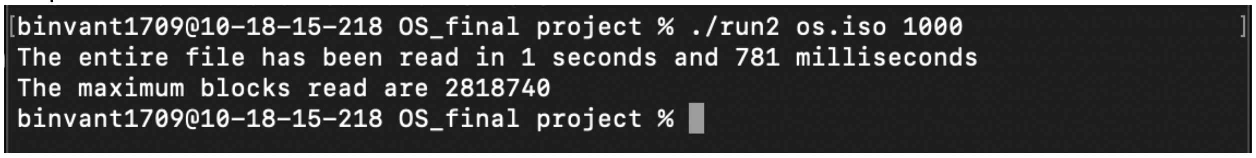
2.)Measurement

Here, we tried our code on the ubuntu image file. The goal here was to find a block count which took a sufficiently long time to read when given a block size. Hence, we tried to find the block count 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 9321078 blocks are read in 5 seconds as per the screenshot.



```
binvant1709@10-18-15-218 OS_final project % ./run2 os.iso 50
The number of blocks read are 9321078 in 5 seconds.
The file size read in 5 seconds is 466053900 bytes
The maximum blocks read are 9321078
binvant1709@10-18-15-218 OS_final project %
```

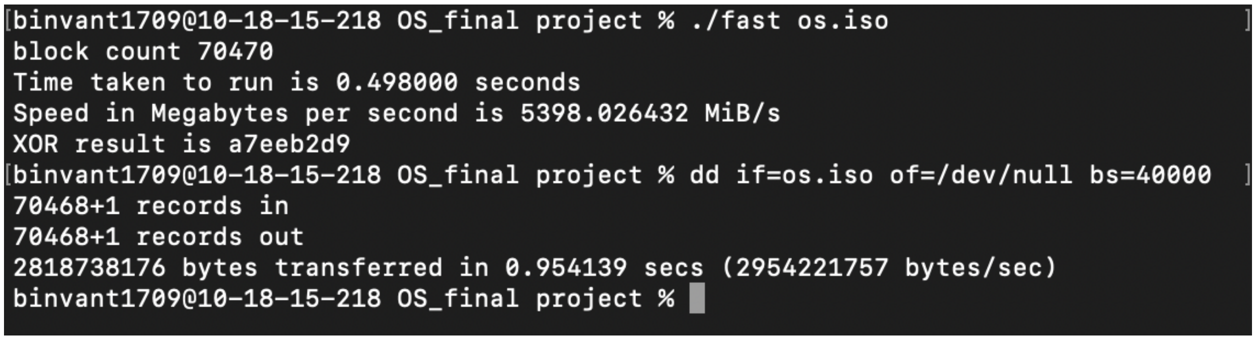
But, if our block size is large enough to read the whole file, we get the following output:



```
binvant1709@10-18-15-218 OS_final project % ./run2 os.iso 1000
The entire file has been read in 1 seconds and 781 milliseconds
The maximum blocks read are 2818740
binvant1709@10-18-15-218 OS_final project %
```

Extra credit idea:

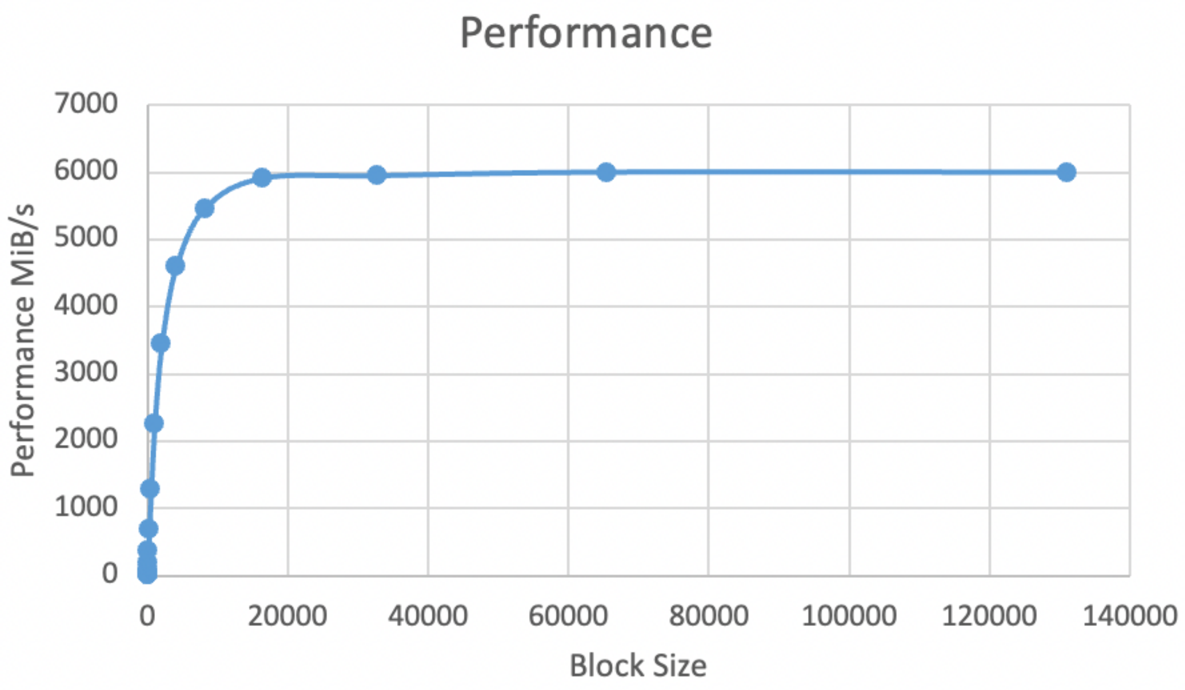
We test our program against the dd command in Linux for the same block size. These were the results:



```
binvant1709@10-18-15-218 OS_final project % ./fast os.iso
block count 70470
Time taken to run is 0.498000 seconds
Speed in Megabytes per second is 5398.026432 MiB/s
XOR result is a7eeb2d9
binvant1709@10-18-15-218 OS_final project % dd if=os.iso of=/dev/null bs=40000
70468+1 records in
70468+1 records out
2818738176 bytes transferred in 0.954139 secs (2954221757 bytes/sec)
binvant1709@10-18-15-218 OS_final project %
```

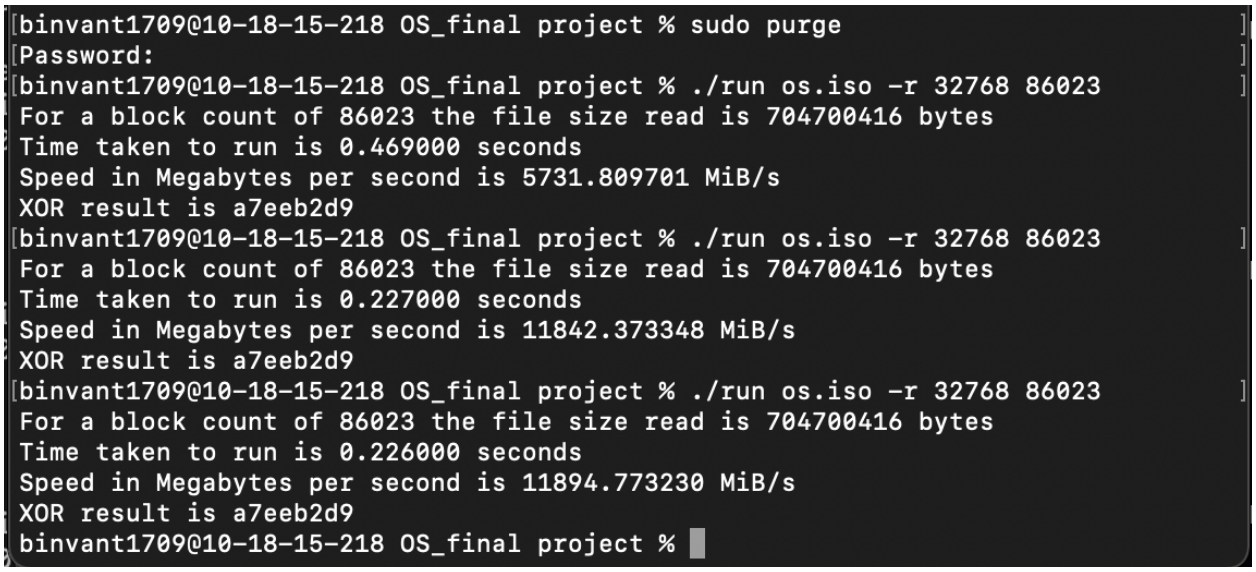
3.)Raw performance

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.



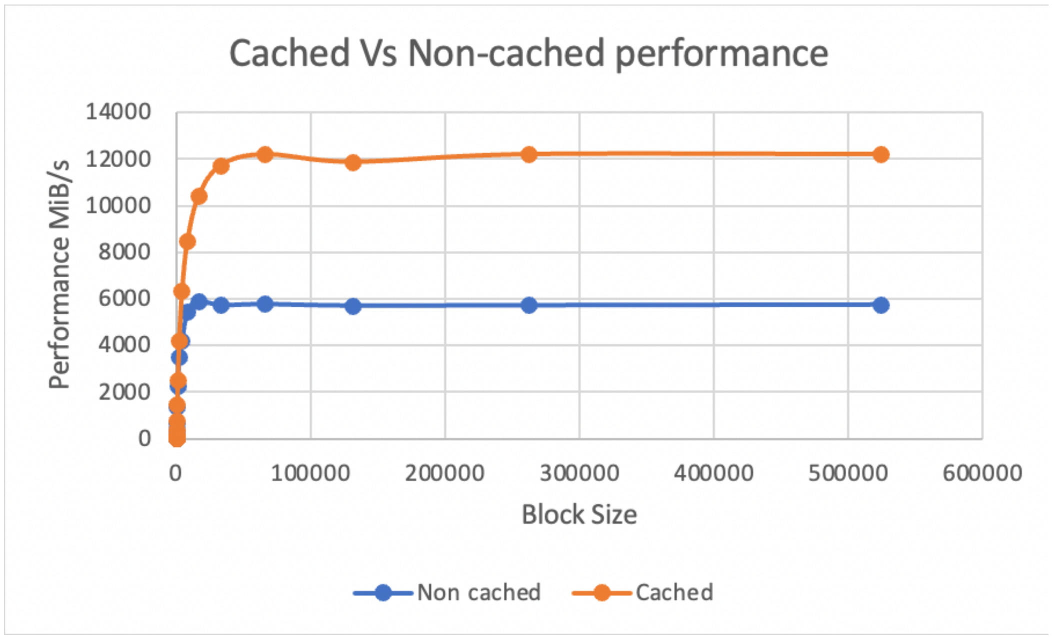
4.)Caching

In the below screenshot, we can see the effect of caching. Here, the run time decreases and performance increases when we keep running the command again and again.
For example, on a significantly large block size of 32768:
After we clear the cache using sudo purge, we get the output of non-cached value and when we keep running it again and again it gets cached and performance increases as shown in the screenshot:



```
binvant1709@10-18-15-218 OS_final project % sudo purge
Password:
binvant1709@10-18-15-218 OS_final project % ./run os.iso -r 32768 86023
For a block count of 86023 the file size read is 704700416 bytes
Time taken to run is 0.469000 seconds
Speed in Megabytes per second is 5731.809701 MiB/s
XOR result is a7eeb2d9
binvant1709@10-18-15-218 OS_final project % ./run os.iso -r 32768 86023
For a block count of 86023 the file size read is 704700416 bytes
Time taken to run is 0.227000 seconds
Speed in Megabytes per second is 11842.373348 MiB/s
XOR result is a7eeb2d9
binvant1709@10-18-15-218 OS_final project % ./run os.iso -r 32768 86023
For a block count of 86023 the file size read is 704700416 bytes
Time taken to run is 0.226000 seconds
Speed in Megabytes per second is 11894.773230 MiB/s
XOR result is a7eeb2d9
binvant1709@10-18-15-218 OS_final project %
```

Here,
The performance increases to 11894.773230 MiB/s from 5731.809701 MiB/s after we keep running it again and again.



The above graph shows the Cached Vs Non cached performance of our program. We varied the block size from 1 to 524288 and in all the cases we can clearly see that cached performance is better than the non-cached performance. We also notice that cached graph plateaus at a much higher performance level compared to non-cached graph.

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

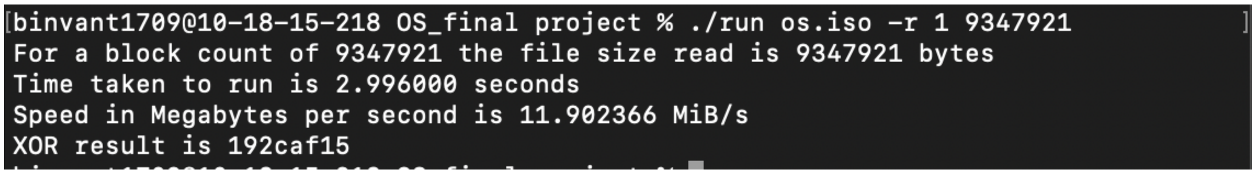
We can use various forms of this command to either free dentries, inodes or page cache.
When we run sudo sh -c "/usr/bin/echo 1 > /proc/sys/vm/drop_cache, Page cache is freed. The page cache could contain any memory mappings to blocks on disk. That could conceivably be buffered I/O, memory mapped files, 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.

5.)System calls

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



```
binvant1709@10-18-15-218 OS_final project % ./run os.iso -r 1 9347921
For a block count of 9347921 the file size read is 9347921 bytes
Time taken to run is 2.996000 seconds
Speed in Megabytes per second is 11.902366 MiB/s
XOR result is 192caf15
```

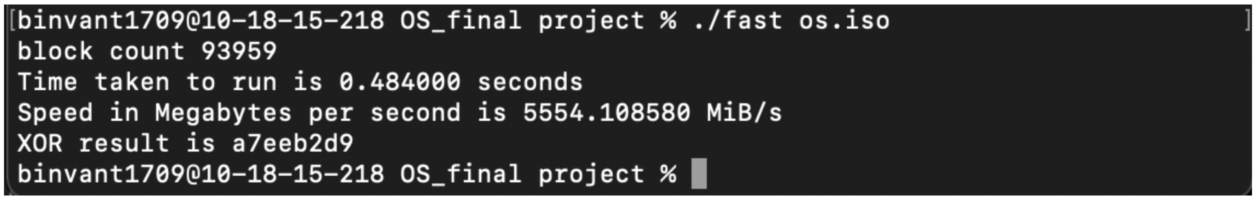
Block count is found using ./run2 script

For this combination of block_count*block_size-
Total blocks read: 9347921
Total time in seconds: 2.996
So, total system calls = blocks read/time = 3120134 system calls/s.

6.)Raw performance

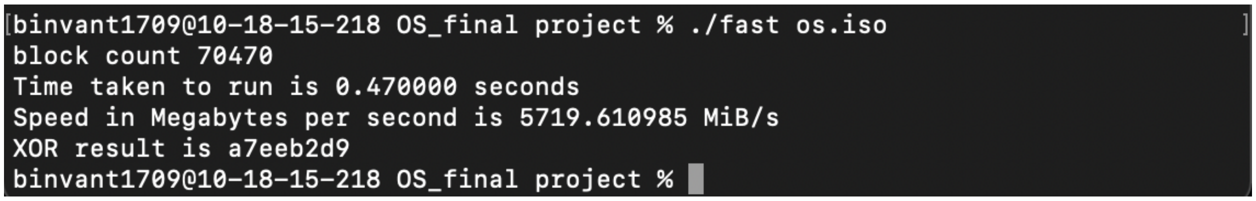
Here, we found that the most optimal block size for our experiment was 30,000. If we go on increasing the block size after that we will see that there is not much improvement in performance and the saturation point is reached.

For block size = 30000:



```
binvant1709@10-18-15-218 OS_final project % ./fast os.iso
block count 93959
Time taken to run is 0.484000 seconds
Speed in Megabytes per second is 5554.108580 MiB/s
XOR result is a7eeb2d9
binvant1709@10-18-15-218 OS_final project %
```

For block size= 40000:

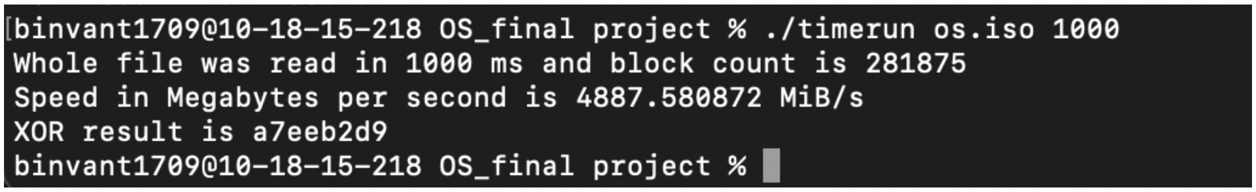


```
binvant1709@10-18-15-218 OS_final project % ./fast os.iso
block count 70470
Time taken to run is 0.470000 seconds
Speed in Megabytes per second is 5719.610985 MiB/s
XOR result is a7eeb2d9
binvant1709@10-18-15-218 OS_final project %
```

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 on our machine.

7.)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.



```
binvant1709@10-18-15-218 OS_final project % ./timerun os.iso 1000
Whole file was read in 1000 ms and block count is 281875
Speed in Megabytes per second is 4887.580872 MiB/s
XOR result is a7eeb2d9
binvant1709@10-18-15-218 OS_final project %
```

Thank you!