# CS-GY 6233 Operating systems - Final Project

| | |
|---|---|
| Date | @December 8, 2021 |
| Team mate 1 | Rigveda Vangipurapu rv2205 |
| Team mate 2 | Jaya Amit Sai Gurrala jg6660 |

## Basics

We have written a program using system calls - open, read, write and close.

The program reads the block_size number of characters each time, and reads block_count number of blocks.

So the total file_size being read by our code is block_size * block_count

Our program returns the XOR value of whatever is read so as to make sure the read operation is taking place correctly.

Usage: ./run <filename> [-r|-w] <block_size> <block_count>

Output:

```
rigvedavangipurapu@Rigvedas-Air OSfinal % gcc run.c -o run
rigvedavangipurapu@Rigvedas-Air OSfinal % ./run dumb.txt -r 1000 10
XOR value: 0000000e
```

💡 Take away from this task: We learnt the usage of these system calls and figured out boiler-plate stuff like how to declare buffers etc

## Measurement

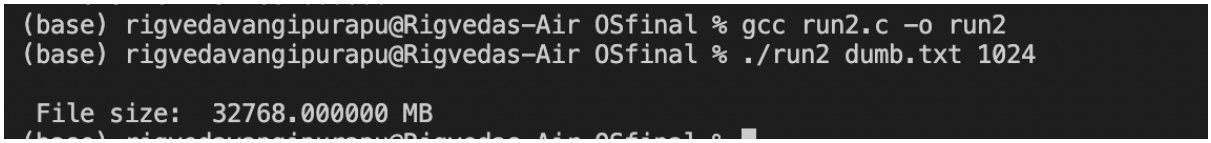
We wrote a program to measure the performance of the program.

Our implementation to the same was as follows:

1. Open the file given as input in the command line.
2. Set timer and block_count to 0.
3. Start reading block_size amount of information from it.
4. Each time one block is read, the time taken is checked.
5. If the cumulative time taken does not exceed 5 seconds, continue the read operation and increment the block_count.
6. If the cumulative time taken reaches (or exceeds) 5 seconds, stop reading.
7. Report the file size to be block_count*block size

Usage: ./run2 <filename> <block_size>

Output:




---

## Extra credit task:

We wrote a program which reads data from a file as blocks, and writes it to another file.

Usage of dd program: dd if=<input_file> of=<output_file>

Usage of our program: ./comparedd <input_file> <output_file> <block_size>

At block size 512, this is how the program compares with the dd program:

```
rigvedavangipurapu@Rigvedas-Air OSfinal % ./comparedd dumb.txt hello.txt 512


file size 1048576 bytes
read: time taken 0.014670 secs
Performance at 512= 71478084.041996 bytes/s
rigvedavangipurapu@Rigvedas-Air OSfinal % dd if=dumb.txt of=hello.txt
2048+0 records in
2048+0 records out
1048576 bytes transferred in 0.011072 secs (94705883 bytes/sec)
```

*There is a minor variation each time the programs are run possibly due to caching and various other factors, this is one of the instances captured for reference.

Running the dd program and our program simultaneously, we noted the following observations:

| File size | dd program: time taken | dd program: performance | our program: time taken | our program: performance |
|---|---|---|---|---|
| 80 bytes | 0.000052 secs | 1539194 bytes/sec | 0.000595 secs | 3442859.555912 bytes/s |
| 1MB | 0.010605 secs | 98874722 bytes/sec | 0.009082 secs | 115455503.927336 bytes/sec |
| 1GB | 17.181206 secs | 164059390 bytes/sec | 18.279118 secs | 154205370.663034 bytes/sec |

## Raw Performance

This program is yet another way to measure the performance of the program.

We witness how the block size affects the performance.

We read the ubuntu-21.04-desktop-amd64.iso file which is sufficiently large to accomodate the block sizes we use.

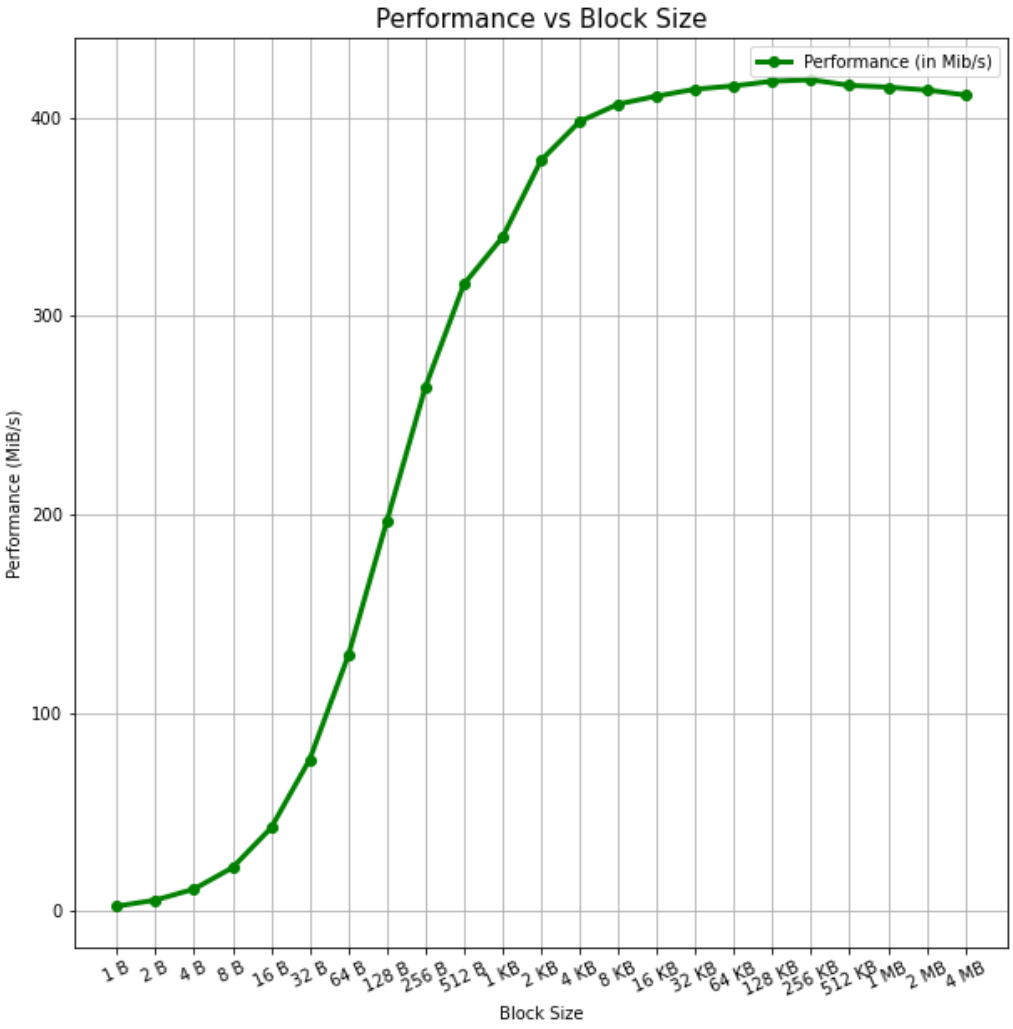
Now we changed the block sizes starting from 1 Byte ranging till 4 MB.

The following values were obtained:

| Block size | Performance (MiB/s) |
|---|---|
| 1 B | 2.675 |

| Block size | Performance (MiB/s) |
|---|---|
| 2 B | 5.754 |
| 4 B | 11.30 |
| 8 B | 22.110 |
| 16 B | 42.153 |
| 32 B | 76.597 |
| 64 B | 128.894 |
| 128 B | 196.524 |
| 256 B | 264.083 |
| 512 B | 315.987 |
| 1 KB | 339.620 |
| 2 KB | 378.327 |
| 4 KB | 397.877 |
| 8 KB | 406.633 |
| 16 KB | 410.70 |
| 32 KB | 414.189 |
| 64 KB | 415.818 |
| 128 KB | 418.186 |
| **256 KB** | **418.896** |
| 512 KB | 416.103 |
| 1 MB | 415.190 |
| 2 MB | 413.76 |
| 4 MB | 411.279 |

Plotting the above values, we obtain the following graph:




Usage: ./performancegraph <block_size>

---

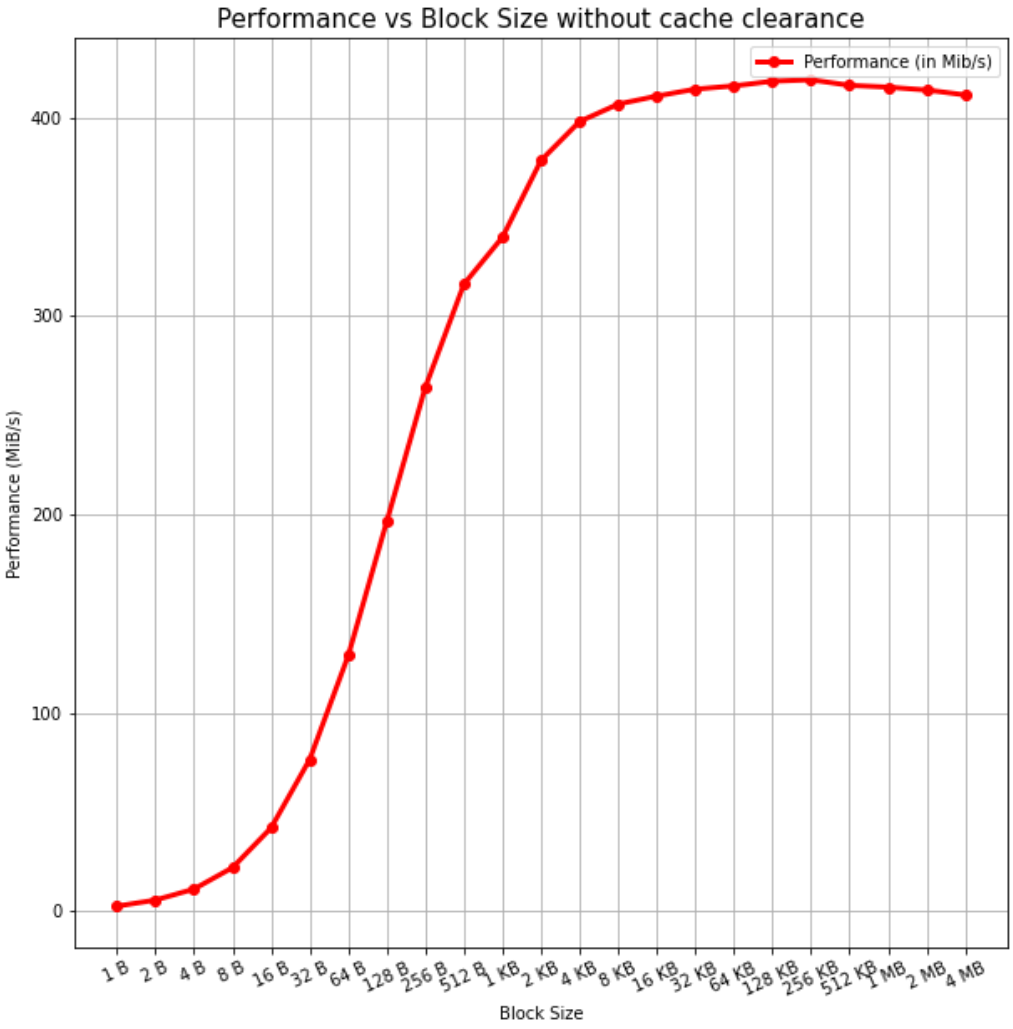
## Caching

We ran the performance measurement code using the same ubuntu-21.04-desktop-amd64.iso file with and without clearing the cache to see the results.

We observed the following results:

Without clearing cache:

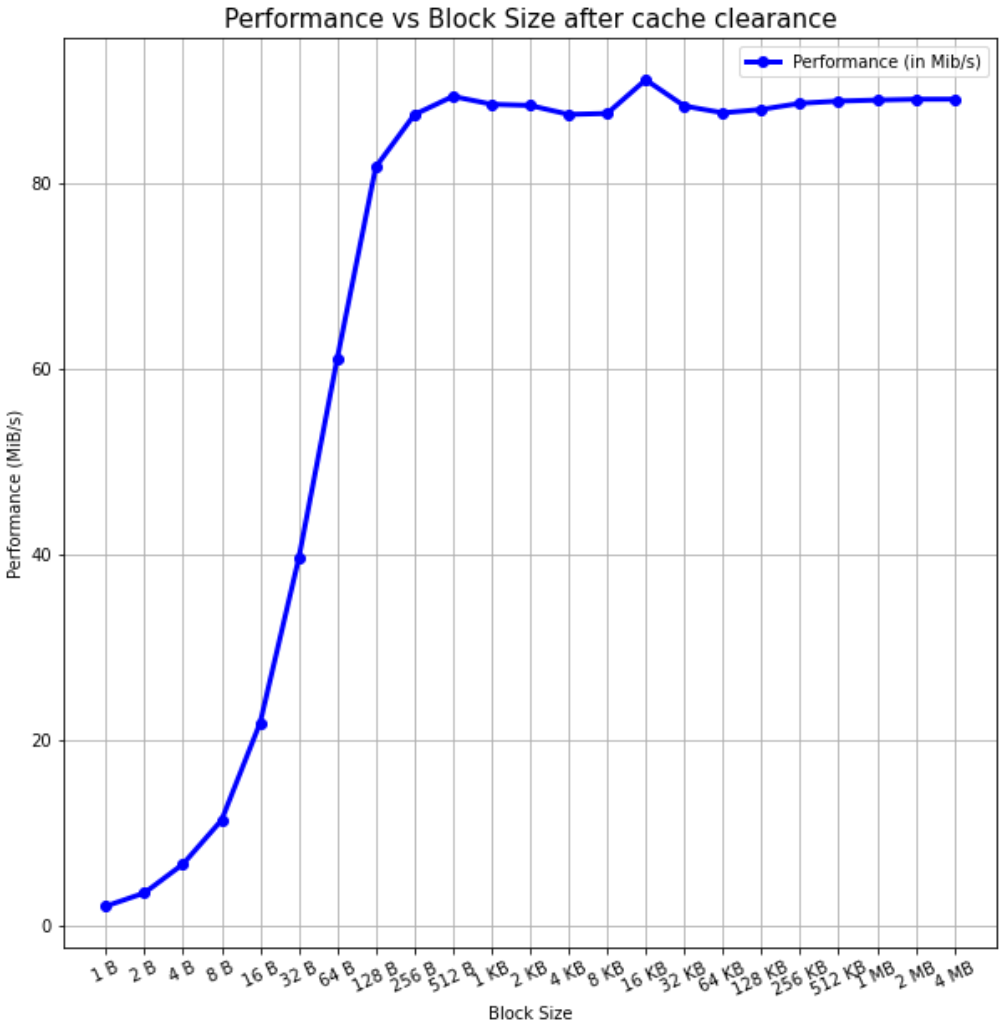
| Block size | Performance (MiB/s) |
|---|---|
| 1 B | 2.675 |
| 2 B | 5.754 |
| 4 B | 11.30 |
| 8 B | 22.110 |
| 16 B | 42.153 |
| 32 B | 76.597 |
| 64 B | 128.894 |
| 128 B | 196.524 |
| 256 B | 264.083 |
| 512 B | 315.987 |
| 1 KB | 339.620 |
| 2 KB | 378.327 |
| 4 KB | 397.877 |
| 8 KB | 406.633 |
| 16 KB | 410.70 |
| 32 KB | 414.189 |
| 64 KB | 415.818 |
| 128 KB | 418.186 |
| **256 KB** | **418.896** |
| 512 KB | 416.103 |
| 1 MB | 415.190 |
| 2 MB | 413.76 |
| 4 MB | 411.279 |




With clearing cache each time:

| Block size | Performance (MiB/s) |
|---|---|
| 1 B | 2.023 |
| 2 B | 3.456 |
| 4 B | 6.567 |
| 8 B | 11.309 |
| 16 B | 21.821 |
| 32 B | 39.532 |
| 64 B | 61.123 |

| Block size | Performance (MiB/s) |
|---|---|
| 128 B | 81.793 |
| 256 B | 87.440 |
| 512 B | 89.413 |
| 1 KB | 88.562 |
| 2 KB | 88.434 |
| 4 KB | 87.464 |
| 8 KB | 87.568 |
| **16 KB** | **91.200** |
| 32 KB | 83.375 |
| 64 KB | 87.633 |
| 128 KB | 88.002 |
| 256 KB | 88.662 |
| 512 KB | 88.901 |
| 1 MB | 89.012 |
| 2 MB | 89.102 |
| 4 MB | 89.110 |




💡 Take away: When the program is executed by clearing the cache, every time the data is fetched from the RAM, resulting in larger read times and lesser performance.
On the other hand, if the data is cached, it is much easier for the CPU to fetch the data and carryout the execution, resulting in smaller read times and greater performance.

---

## Extra Credit

```
sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"
```

There are 3 options to clear the cache without interrupting any processes or services:

1. echo 1 : to clear page cache only
2. echo 2 : to clear dentries and inodes
3. echo 3 : to clear page cache, dentries and inodes

An **Inode** is a data structure which provides a representation of a file.

**Dentries** is a data structure which represents a directory or a folder. So, dentries can be used to store cache, if they exist then store or check from memory.

The **Page cache** is anything the OS could store in the memory or hold it rather read it from a file.

---

## System Calls

Keeping block size as 1 Byte, we tried various system calls to see how many system calls can be performed in a second.

We experimented with the following system calls:

read

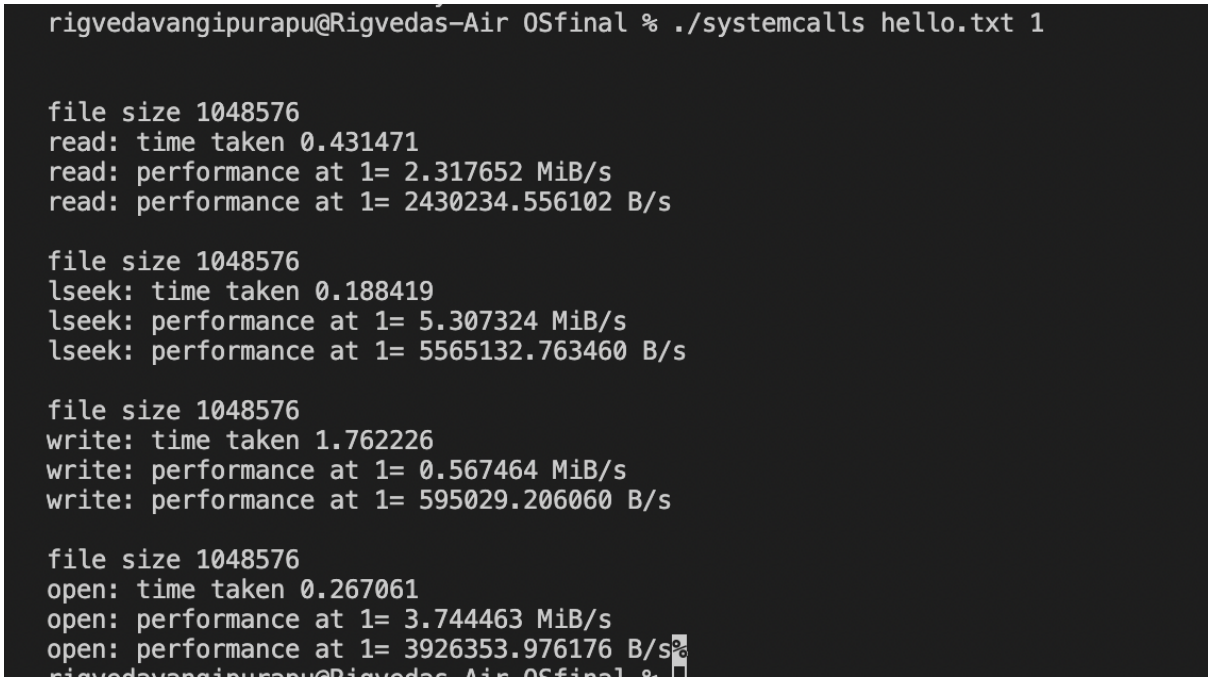
write

open

lseek

| System call | File Size | Time taken (secs) | Performance (B/s) (Number of system calls per second) | Performance (MiB/s) |
|---|---|---|---|---|
| read | 1MB | 0.431471 | 2430234.556102 B/s | 2.317652 MiB/s |
| write | 1MB | 1.762226 | 595029.206060 B/s | 0.567464 MiB/s |
| open | 1MB | 0.267061 | 3926353.976176 B/s | 3.744463 MiB/s |
| lseek | 1MB | 0.188419 | 5565132.763460 B/s | 5.307324 MiB/s |

Output:




> 💡 Take away from this task: We observed that the number of system calls needed for block size=1 is very high, which means a very large number of I/O operations.
> Since I/O operations are costly, the time taken to read a file is very large given block size = 1

---

## Raw Performance

Based on the previous observations, we chose our block size to be: **256KB**

Usage: ./fast <file name>

Output:

```
(base) rigvedavangipurapu@Rigvedas-Air OSfinal % gcc fast.c -o fast
(base) rigvedavangipurapu@Rigvedas-Air OSfinal % ./fast ubuntu-21.04-desktop-amd64.iso
XOR value is a7eeb2d9
Time taken to run 262144 is 6.545756

 Block count = 10753
 Speed = 410.685955 MiB/s
(base) rigvedavangipurapu@Rigvedas-Air OSfinal %
```

## Expected performance:

| Block size: 256 KB | Performance (MiB/s) |
|---|---|
| Without clearing cache | 418.896 |
| After clearing cache | 88.662 |