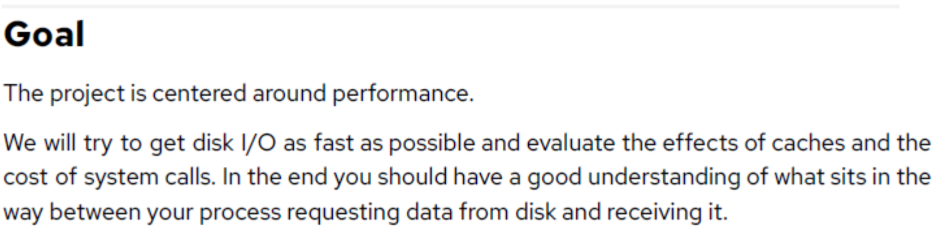
Final Project - OS

CS-GY 6233 - Fall 2023

Submitted by:

Pranav Mohril - N13784838

Venkata Naga Dheeraj Pavuluri - N16788493



Goal

The project is centered around performance.

We will try to get disk I/O as fast as possible and evaluate the effects of caches and the cost of system calls. In the end you should have a good understanding of what sits in the way between your process requesting data from disk and receiving it.

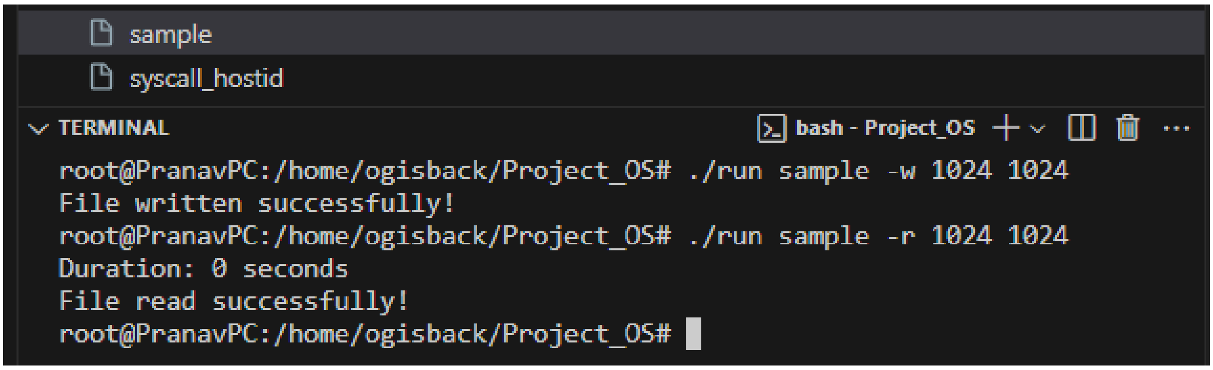
Important! Please make sure you have superuser access while running the script and code files as caching command (embedded in code) requires root access.

Part 1: Basics

The code for 1.Basics is in `run.cpp`

Format to run: `./run <filename> [-r|-w] <block_size> <block_count>`

Examples:



```
root@PranavPC:/home/ogisback/Project_OS# ./run sample -w 1024 1024
File written successfully!
root@PranavPC:/home/ogisback/Project_OS# ./run sample -r 1024 1024
Duration: 0 seconds
File read successfully!
root@PranavPC:/home/ogisback/Project_OS#
```

Part 2 and Part 3: Performance (MiB/s) Graph for different block sizes

Using `run2.cpp`, we are able to return the block count for each file and then use it find a file that reads in reasonable time

Usage: `./run2.cpp <filename> <block_size>`

By trial and error, we were able to find a file that would be read in reasonable time → test128 (2.00 GB)

We use `raw_performance.cpp` to find the performance in MiB/s for the following block sizes.

Usage: `./raw_performance <filename> <block_size>`

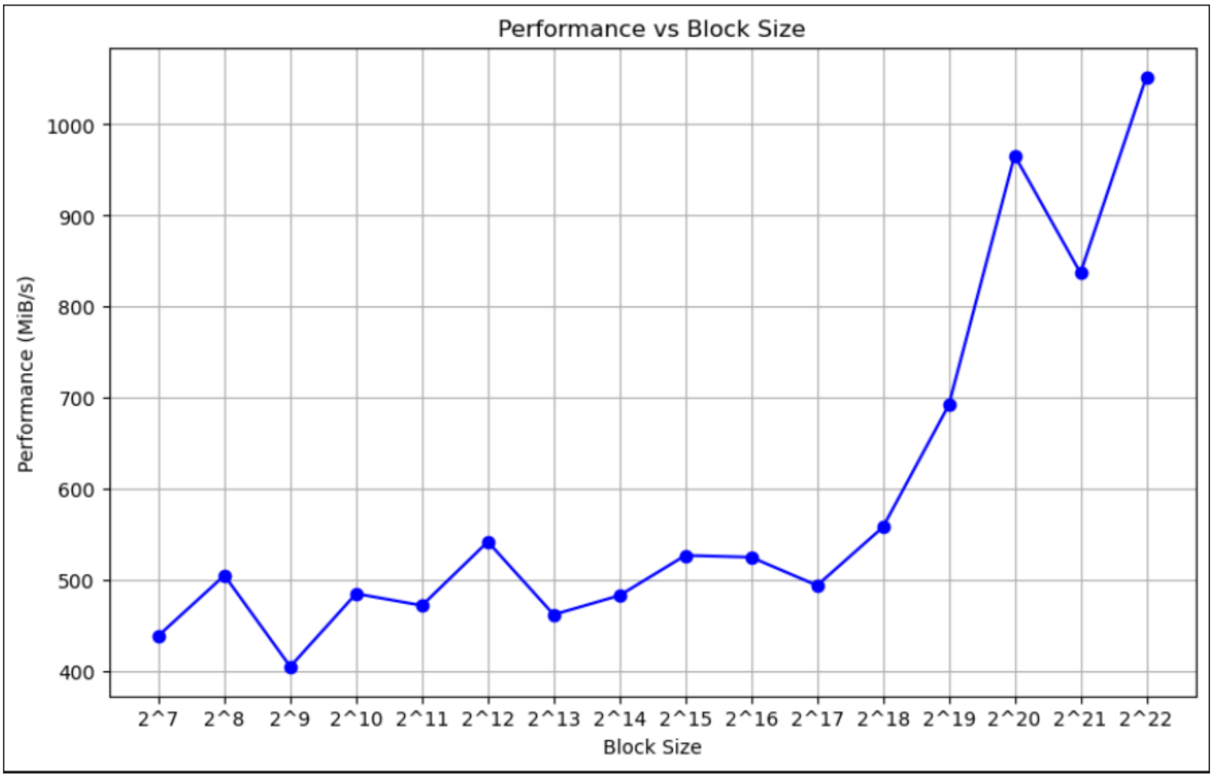
** While measuring the performance, we cleared the cache after every call to ensure accurate results.*

Block Sizes

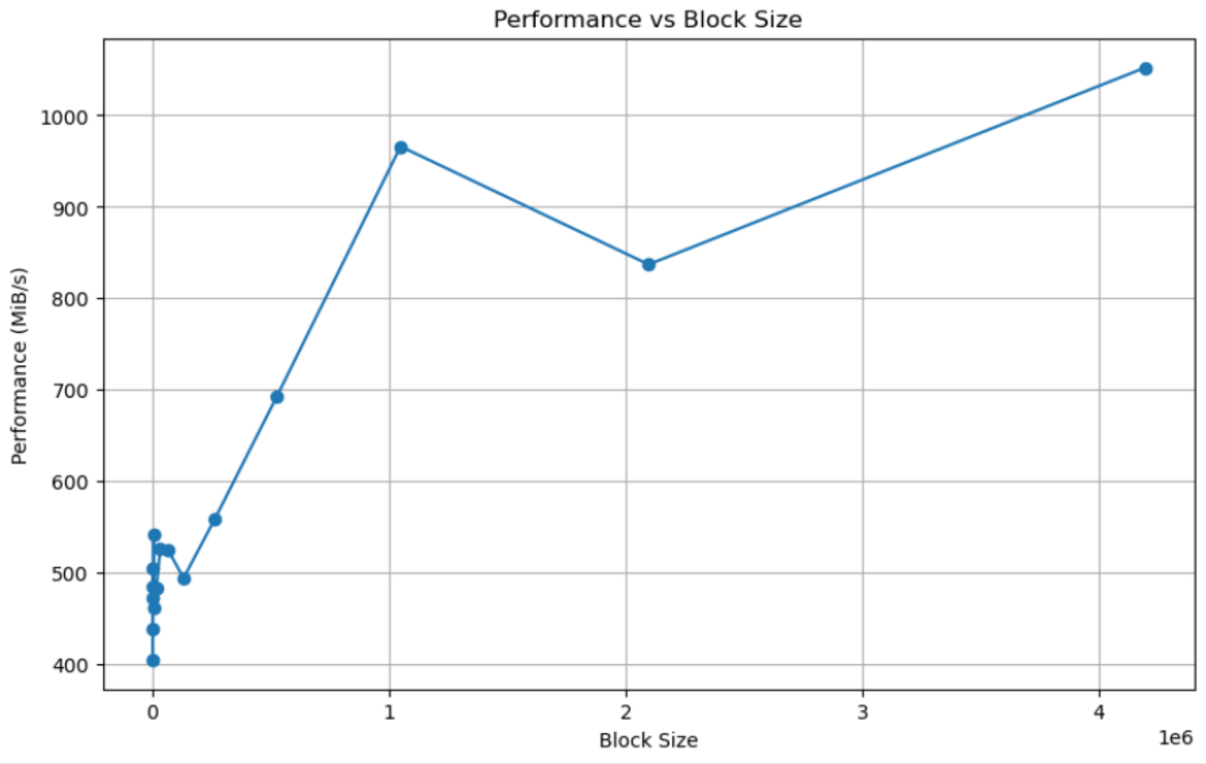
[128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304]

Performance (MiB/s)

[439, 505, 405, 485, 472, 542, 462, 483, 527, 525, 494, 558, 692, 966, 837, 1052]



Block sizes placed at same distance on x axis



Spatially correct (X axis points are multiple of 10^6)

The “dd” command

The “dd” (disk/data duplicator or infamously, disk destroyer) command is a linux utility that allows us to copy data from one source to another. Unlike “cp” command which copies individual files, “dd” is used to read and write to block devices like hard drives.

In shell: `> dd if=input_file of=output_file bs=block_size`

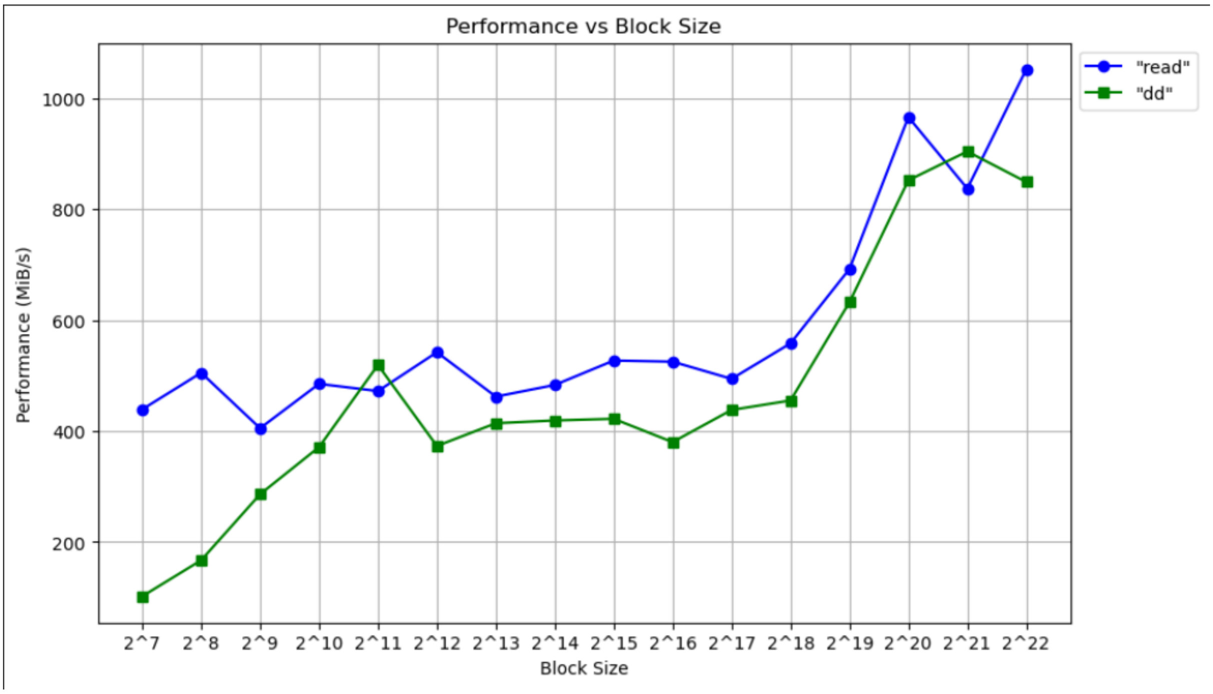
Example usage: `dd if=test128 of=/dev/null bs=1048576`

Block Sizes

[128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304]

Performance of “dd” in MiB/s (No Caching)

[102, 167, 286, 371, 519, 373, 414, 419, 422, 380, 438, 455, 632, 852, 904, 849]



** The above experiment uses “dd” to read from test128 and write it to /dev/null where it is discarded. The performance is slightly underwhelming but I think it due to the fact that the optimal block size has not been explored yet. (Probably around 16 MB - sources)*

Part 4: Performance (MiB/s) - Caching vs No Caching

We can use `caching.cpp` to generate all the results by running the file only once. Caching has been taken care of in the code itself.

Usage: `./caching <filename>`

Command to drop cache: `sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"`

Block Sizes

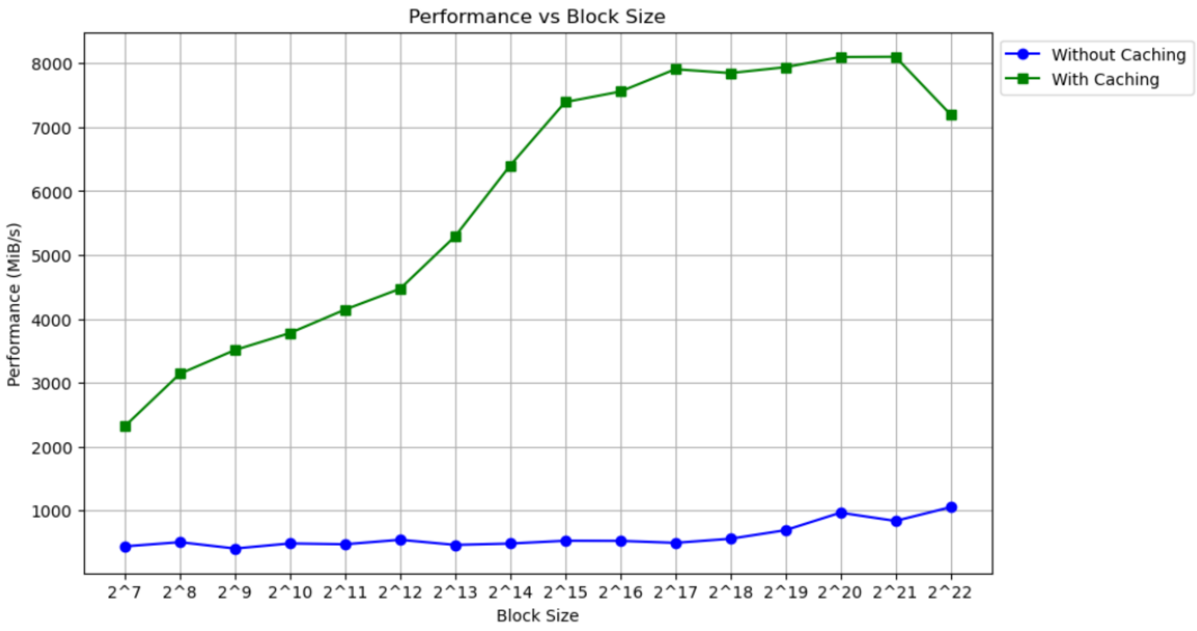
[128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304]

Performance Without Caching

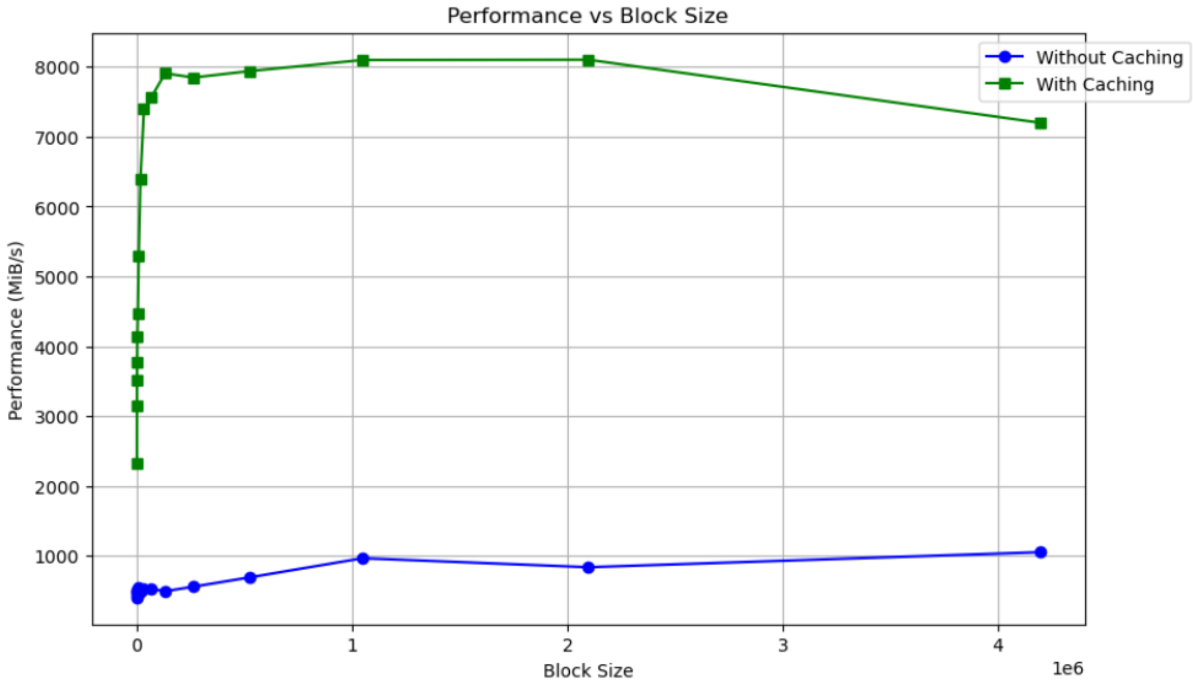
[439, 505, 405, 485, 472, 542, 462, 483, 527, 525, 494, 558, 692, 966, 837, 1052]

Performance With Caching

[2321, 3141, 3512, 3778, 4145, 4471, 5292, 6400, 7393, 7557, 7907, 7847, 7939, 8098, 8102, 7200]



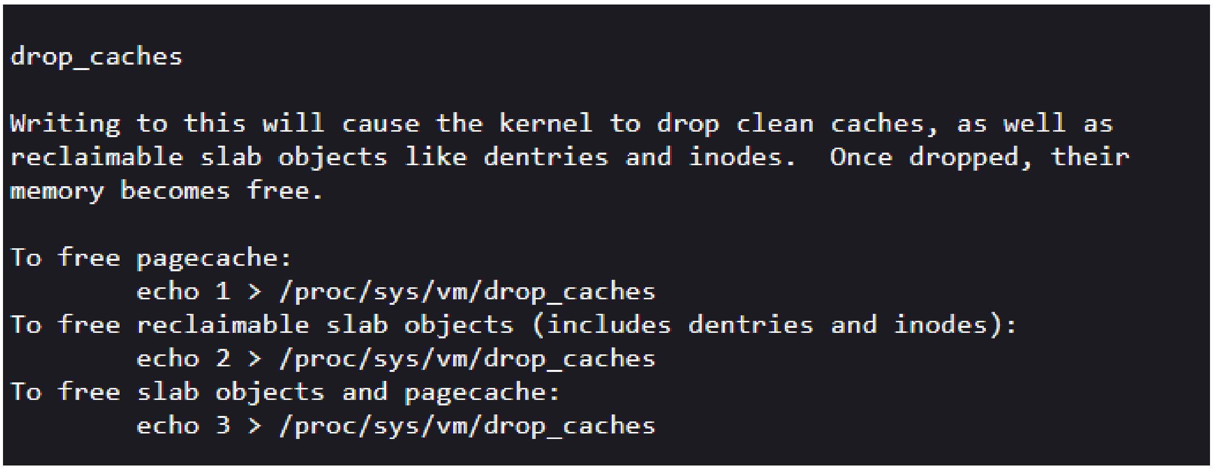
Block Sizes equidistant



Block sizes spatially correct

Extra Credit: Why “3”

In linux, writing values to `/proc/sys/vm/drop_caches` is a way to clear different types of caches. (Source: Kernel Docs)



```
drop_caches

Writing to this will cause the kernel to drop clean caches, as well as
reclaimable slab objects like dentries and inodes.  Once dropped, their
memory becomes free.

To free pagecache:
        echo 1 > /proc/sys/vm/drop_caches
To free reclaimable slab objects (includes dentries and inodes):
        echo 2 > /proc/sys/vm/drop_caches
To free slab objects and pagecache:
        echo 3 > /proc/sys/vm/drop_caches
```

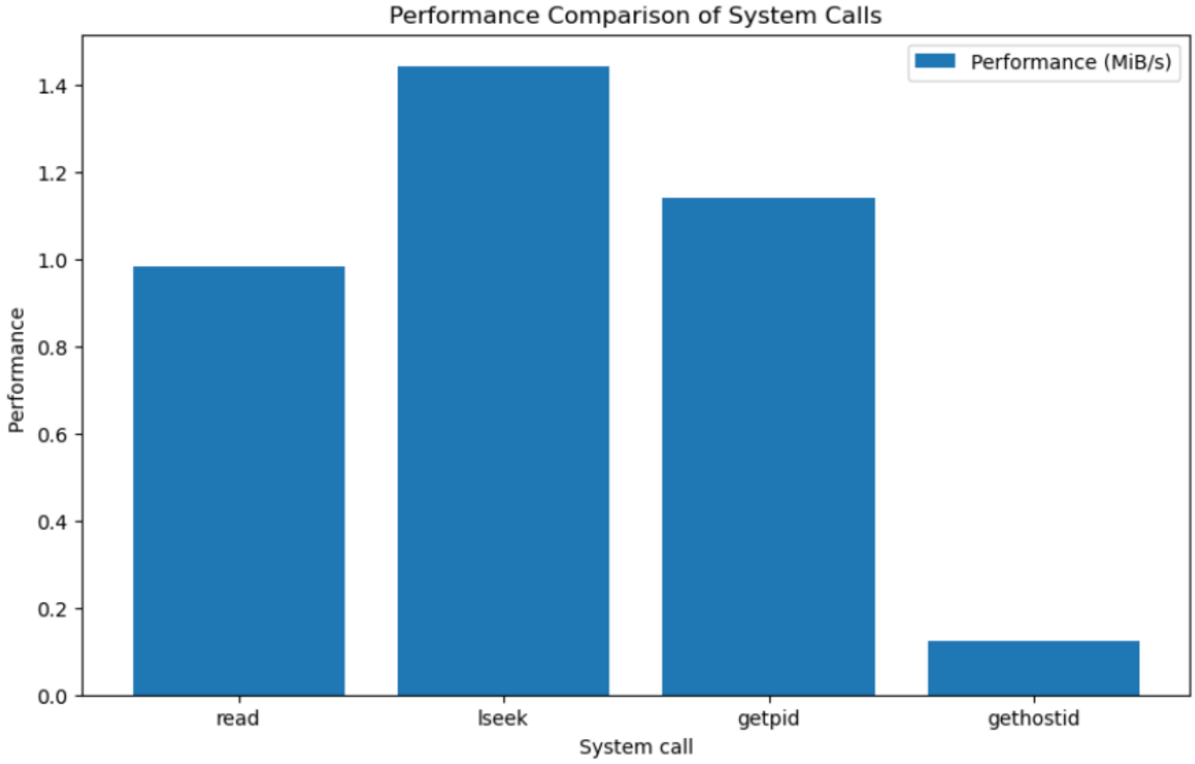
ref: Kernel Docs

Here, using 3 will drop both “pagecache” and “free slab objects”.

Part 5: System Calls

We measure the performance for *10 seconds* using *block size = 1* to draw accurate observations. (10 seconds ⇒ as the block size is extremely small, reading the entire file takes too long)

syscall	Performance (MiB/s)	Equivalent Syscalls (B/s)
read	0.982676	1.03041e+06
lseek	1.44276	1.51284e+06
getpid	1.13915	1.19449e+06
gethostid	0.124913	130980



** The graph for syscalls/s would be similar to the one above (every value is multiplied by 1024*1024)*

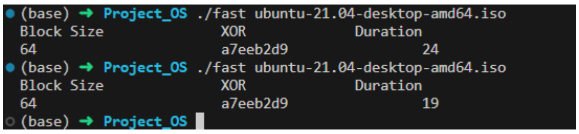
From the graph, we can observe that performance depends on how cpu- intensive the syscall is. Also, most of the time is spent trapping in the kernel due to the very small block size.

Part 6: Raw Performance

To maximize read performance, we will utilize multithreading to read multiple blocks based on `threadId*blockSize` *by utilizing* `seekg`*. We wait for all threads to complete their work using* `join` *and then finally XOR the results from the threads. As* `XOR` *is a deterministic function, splitting up the work into threads gives a performance boost. We are XOR'ing 4 bytes of data at once.*

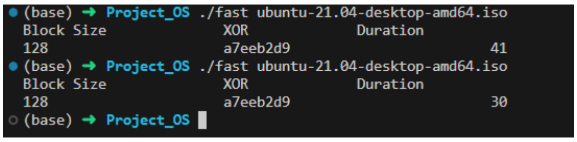
We use `fast.cpp` with multiple file sizes to generate results and check the most optimal block size.

Observations: No. of Threads = 4



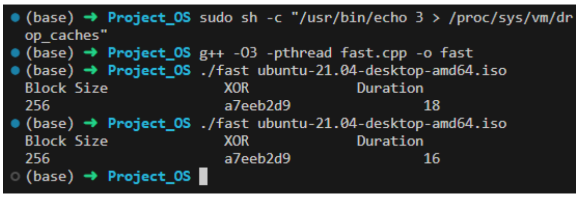
```
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  64                       a7eeb2d9              24
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  64                       a7eeb2d9              19
○ (base) ➜ Project_OS ▌
```

BS: 64, -O3, Non-Caching vs Caching



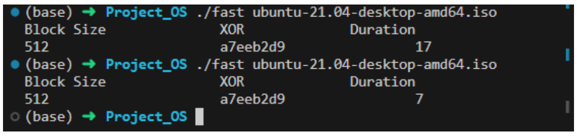
```
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  128                      a7eeb2d9              41
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  128                      a7eeb2d9              30
○ (base) ➜ Project_OS ▌
```

BS: 128, -O3, Non-Caching vs Caching



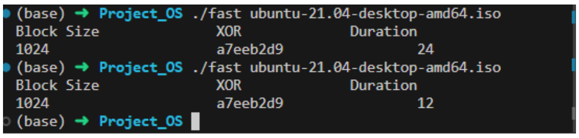
```
● (base) ➜ Project_OS sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/dr
  op_caches"
● (base) ➜ Project_OS g++ -O3 -pthread fast.cpp -o fast
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  256                      a7eeb2d9              18
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  256                      a7eeb2d9              16
○ (base) ➜ Project_OS ▌
```

BS: 256, -O3, Non-Caching vs Caching



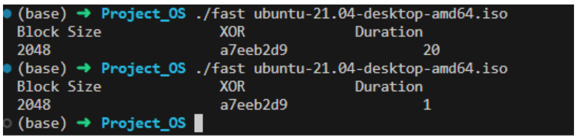
```
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  512                      a7eeb2d9              17
● (base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size               XOR            Duration
  512                      a7eeb2d9               7
○ (base) ➜ Project_OS ▌
```

BS: 512, -O3, Non-Caching vs Caching



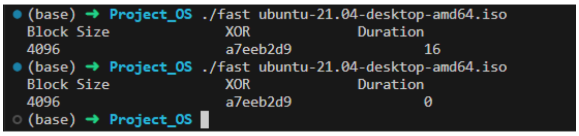
```
(base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size               XOR            Duration
1024                     a7eeb2d9              24
(base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size               XOR            Duration
1024                     a7eeb2d9              12
(base) ➜ Project_OS ▌
```

BS: 1024, -O3, Non-Caching vs Caching



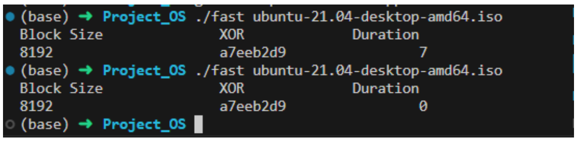
```
(base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size               XOR            Duration
2048                     a7eeb2d9              20
(base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size               XOR            Duration
2048                     a7eeb2d9               1
(base) ➜ Project_OS ▌
```

BS: 2048, -O3, Non-Caching vs Caching



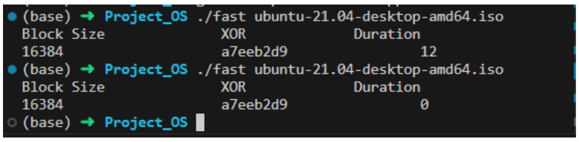
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  4096                    a7eeb2d9              16
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  4096                    a7eeb2d9              0
○ (base) → Project_OS
```

BS: 4096, -O3, Non-Caching vs Caching



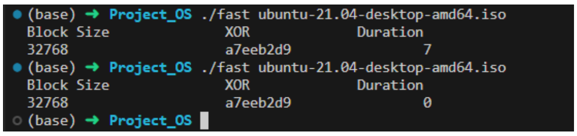
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  8192                    a7eeb2d9              7
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  8192                    a7eeb2d9              0
○ (base) → Project_OS
```

BS: 8192, -O3, Non-Caching vs Caching



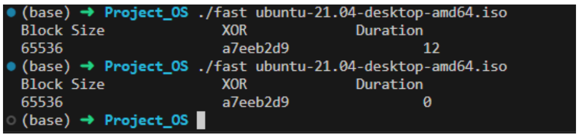
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  16384                   a7eeb2d9              12
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  16384                   a7eeb2d9              0
○ (base) → Project_OS
```

BS: 16384, -O3, Non-Caching vs Caching



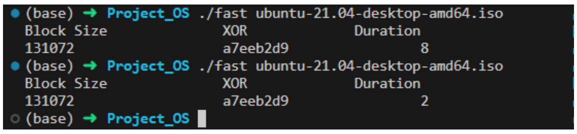
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  32768                   a7eeb2d9              7
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  32768                   a7eeb2d9              0
○ (base) → Project_OS
```

BS: 32768, -O3, Non-Caching vs Caching



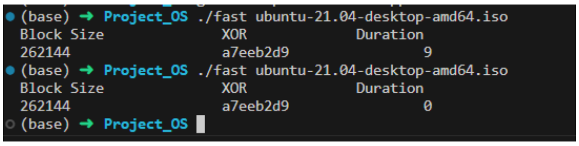
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  65536                   a7eeb2d9              12
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  65536                   a7eeb2d9              0
○ (base) → Project_OS
```

BS: 65536, -O3, Non-Caching vs Caching



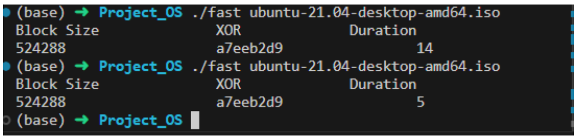
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  131072                  a7eeb2d9              8
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  131072                  a7eeb2d9              2
○ (base) → Project_OS
```

BS: 131072, -O3, Non-Caching vs Caching



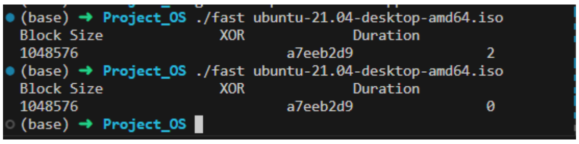
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  262144                  a7eeb2d9              9
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  262144                  a7eeb2d9              0
○ (base) → Project_OS
```

BS: 262144, -O3, Non-Caching vs Caching



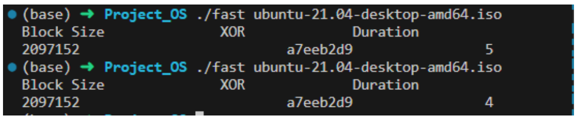
```
(base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size              XOR             Duration
524288                  a7eeb2d9              14
(base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size              XOR             Duration
524288                  a7eeb2d9              5
(base) → Project_OS
```

BS: 524288, -O3, Non-Caching vs Caching



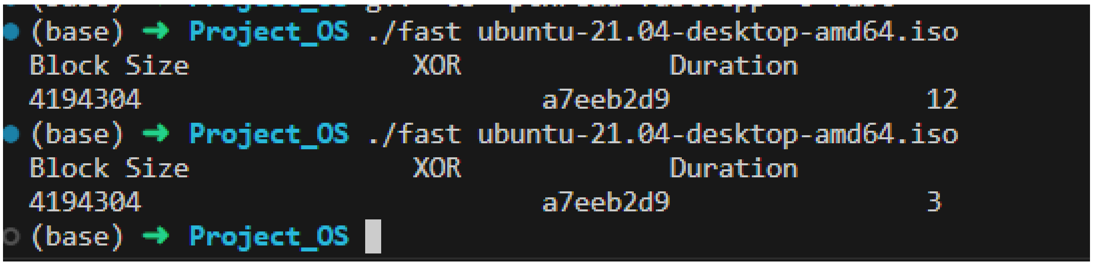
```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  1048576                         a7eeb2d9              2
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  1048576                         a7eeb2d9              0
○ (base) → Project_OS
```

BS: 1048576, -O3, Non-Caching vs Caching



```
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  2097152                         a7eeb2d9              5
● (base) → Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
  Block Size              XOR             Duration
  2097152                         a7eeb2d9              4
```

BS: 2097152, -O3, Non-Caching vs Caching



```
(base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size              XOR             Duration
4194304                         a7eeb2d9                12
(base) ➜ Project_OS ./fast ubuntu-21.04-desktop-amd64.iso
Block Size              XOR             Duration
4194304                         a7eeb2d9                3
(base) ➜ Project_OS
```

BS: 4194304, -O3, Non-Caching vs Caching

No. of Threads ⇒ 4

Optimal Block size: 1048576 ⇒ 1024*1024 ⇒ 1 MB (2 seconds)

Block Size	Time(Non-Cached)	Time(Cached)	XOR Value
64	24	19	a7eeb2d9
128	41	30	a7eeb2d9
256	18	16	a7eeb2d9
512	17	7	a7eeb2d9
1024	24	12	a7eeb2d9
2048	20	1	a7eeb2d9
4096	16	0	a7eeb2d9
8192	7	0	a7eeb2d9
16384	12	0	a7eeb2d9
32768	7	0	a7eeb2d9
65536	12	0	a7eeb2d9
131072	8	2	a7eeb2d9
262144	9	0	a7eeb2d9
524288	14	5	a7eeb2d9
1048576	2	0	a7eeb2d9
2097152	5	4	a7eeb2d9
4194304	12	3	a7eeb2d9

** We have modified the `fast.cpp` to have the optimal block size by default. For grading purposes, you can directly run the file as `./fast <filename>`.*