CS-GY 6233 Operating Systems

Final Project

Team Details:

Student 1:
Name: Aniket Lavjibhai Saliya
Net ID: as15858
S - ID: N12508148

Student 2:
Name: Archilkumar Rameshkumar Chovatiya
Net ID: ac9137
S - ID: N17815964

Professor: Kamen Yotov

CS-GY 6233 Operating Systems

Final Project

System environment: Ubuntu Linux.
Run command ‘./build’ or ‘make’ to compile files.

1. Basic

Way to execute: ./run <filename> [-r|-w] <block_size> <block_count>
file size will be “block_size*block_count*4” bytes.

```
ubuntu@ubuntu:~/Desktop/OS$ ./run ubuntu.iso -r 1024 10000
File size in bytes: 40960000
Total execution time in seconds: 0.052000
ubuntu@ubuntu:~/Desktop/OS$ ./run small.bin -w 1024 10000
Total bytes written in the file: 40960000
Total execution time in seconds: 0.037000
```

2. Measurement

Way to execute: ./run2 <filename> <block_size>
To find a reasonable file size, we are measuring read time for block_counts 1,2,4,8,...,2^X only. Using this approach, if reasonable block_count is N then the program will only attempt to measure read time for log(N) time which will help to avoid unnecessary computation.

Extra Credit: learn about the `dd` program in Linux and see how your program's performance compares to it!

```
ubuntu@ubuntu:~/Desktop/OS$ sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"
ubuntu@ubuntu:~/Desktop/OS$ dd if=ubuntu.iso of=/dev/null bs=4194304
672+1 records in
672+1 records out
2818738176 bytes (2.8 GB, 2.6 GiB) copied, 2.00621 s, 1.4 GB/s
ubuntu@ubuntu:~/Desktop/OS$ dd if=ubuntu.iso of=/dev/null bs=4194304
672+1 records in
672+1 records out
2818738176 bytes (2.8 GB, 2.6 GiB) copied, 0.428847 s, 6.6 GB/s
```

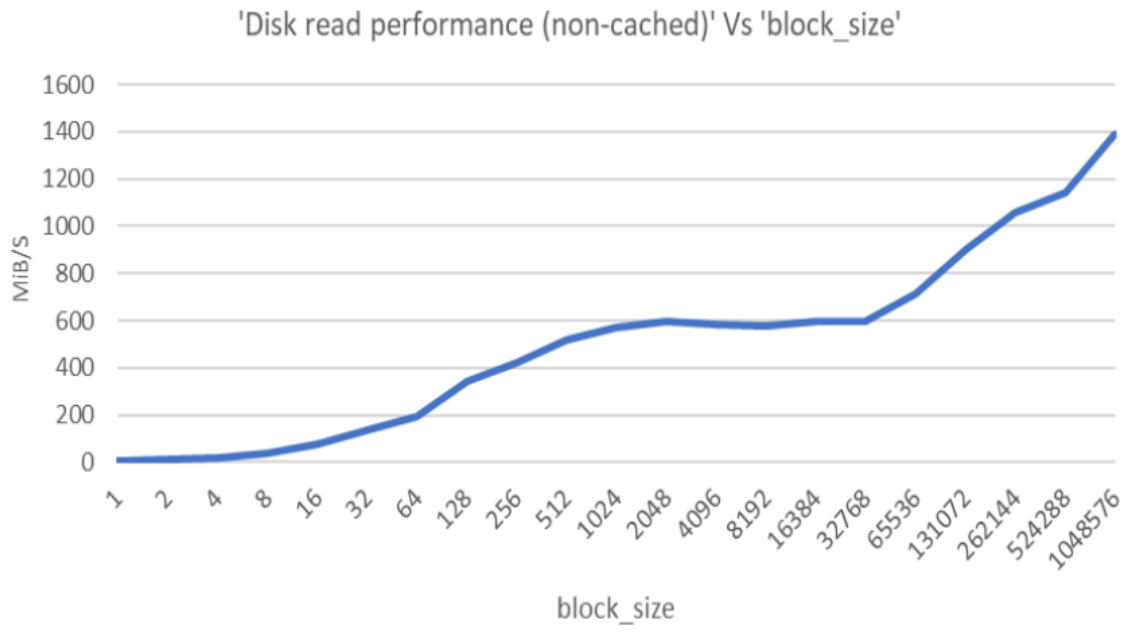
- When we read the file with **“dd”** command, our non-cached disk read speed is 1.4 GB/s. which is very close to our non-cached disk speed 1.3 GB/s for chosen block_size (1048576) .
- But once caches created, when I perform read operation on file with **“dd”** command, my disk speed is 6.6 GB/s whereas my cached disk read speed is 2.7 GB/s for chosen block_size (1048576).

Final Project

3. Raw Performance

Non-Cached Perfomance data and Graph:

block_size	non-cached performance (MiB/s)
1	4.919
2	9.666
4	19.136
8	37.831
16	73.988
32	137.487
64	191.384
128	343.97
256	422.181
512	515.415
1024	568.1
2048	598.044
4096	584.976
8192	579.022
16384	594.312
32768	599.619
65536	712.844
131072	899.133
262144	1055.534
524288	1142.22
1048576	1386.616



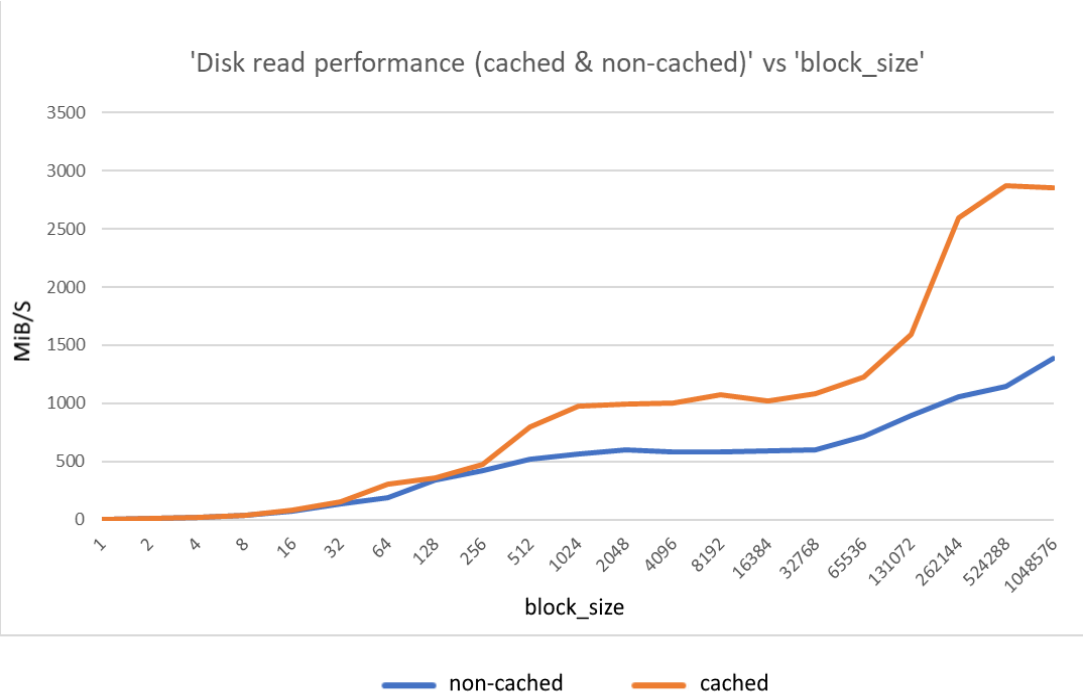
CS-GY 6233 Operating Systems

Final Project

4. Caching

Cached performance data and Graph:

block_size	non-cached performance (MiB/s)	cached performance (MiB/s)
1	4.919	4.977
2	9.666	9.768
4	19.136	19.581
8	37.831	38.514
16	73.988	78.467
32	137.487	154.008
64	191.384	303.003
128	343.97	355.617
256	422.181	470.94
512	515.415	792.723
1024	568.1	971.422
2048	598.044	997.443
4096	584.976	1005.77
8192	579.022	1076.195
16384	594.312	1018.019
32768	599.619	1078.746
65536	712.844	1222.504
131072	899.133	1592.61
262144	1055.534	2599.428
524288	1142.22	2873.563
1048576	1386.616	2856.327



CS-GY 6233 Operating Systems

Final Project

Observation: Till block size 256 performance of cached and non cached read is simmiler. After 256 block size we can observe noticeable performance increase in cached read in compare to non-cached read. After 131072 block size performance of cached read increases drastically in compare to non-cached read.

Extra Credit: Why "3"? Read up on it and explain.
We are using 3 here to clear all kind of caches in linux system. But first let's understand different kind of caches in linux.
There are three forms of caches:
Page Cache
Dentry Cache
inode Cache
PageCache is a cached file. Recently accessed files are stored here, so you don't need to query again from disk unless the file is modified or the cache is cleared to make room for other data. This reduces reads and writes to the disk and allows files to be read from RAM much faster, resulting in faster speed.

Dentry and inode caches are directories and file attributes. This information is closely related to PageCache, but it does not actually contain the actual contents of the file. This cache also reduces hard drive I/O operations.

If we want clear only Page Cache then use (echo 1)
Full Command: sudo sh -c /usr/bin/echo 1 > /proc/sys/vm/drop_caches

If we want to clear Dentry and inode caches then use (echo 2)
Full command: sudo sh -c /usr/bin/echo 2 > /proc/sys/vm/drop_caches

If we want clear all caches(PageCache, Dentry and inode) then use (echo 3)
Full Command: sudo sh -c /usr/bin/echo 3 > /proc/sys/vm/drop_caches

CS-GY 6233 Operating Systems

Final Project

5. System Calls

```
ubuntu@ubuntu:~/Desktop/OS$ sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"
ubuntu@ubuntu:~/Desktop/OS$ ./run2 10GB.bin 1
Block size: 1
Block count: 8388608
File size in bytes: 33554432
Total execution time in seconds: 6.841000
Read speed in B/s: 4904902
Read speed in MiB/s: 4.678
ubuntu@ubuntu:~/Desktop/OS$ ./run3 10GB.bin
Systemcall(lseek) speed in systemcalls/s: 6403518
```

Way to execute: ./run3 <filename>

- When Buffer size(Block_size) is 1, read systemcall make 4904902 system calls per second.
- Here we have made an additional file run3.c which is used to measure speed of “lseek” systemcall.
- As lseek systemcall does less work than read systemcall. Here, lseek systemcall make 6403518 systemcalls per second which is more than read systemcall.

6. Raw Performance

Way to execute: ./fast <filename>

With XOR Operation:

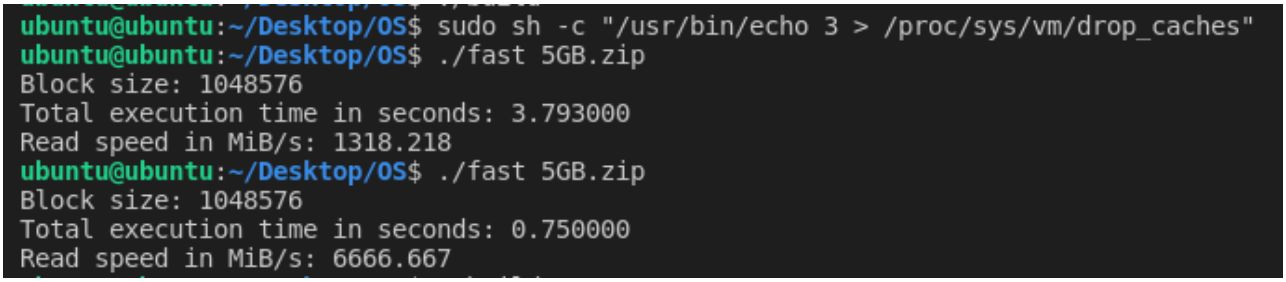
```
ubuntu@ubuntu:~/Desktop/OS$ sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"
ubuntu@ubuntu:~/Desktop/OS$ ./fast 5GB.zip
XOR: 78e0886b
Block size: 1048576
Total execution time in seconds: 5.784000
Read speed in MiB/s: 864.454
ubuntu@ubuntu:~/Desktop/OS$ ./fast 5GB.zip
XOR: 78e0886b
Block size: 1048576
Total execution time in seconds: 3.417000
Read speed in MiB/s: 1463.272
```

- Here 5GB.zip file is read in **5.784 seconds** with read speed 864.454 MiB/s which is non-cache performance (with XOR operation).
- After caching, 5GB.zip is file read in **3.417 seconds** with disk speed **1463.272 MiB/s** (with XOR operation).

CS-GY 6233 Operating Systems

Final Project

Without XOR Operation:



- Here 5GB.zip file is read in **3.793 seconds** with read speed **1318.218 MiB/s** which is non-cache performance (without XOR operation).
- After caching, 5GB.zip is file read in **0.75 seconds** with disk speed **6666.667 MiB/s** (without XOR operation).

Observation: XOR operation takes significant time along with reading operation so that, observed read speed for cached and non-cached performance measured while performing XOR operation will be very much less than reading speed for cached and non-cached performance without performing XOR operation.

What we have done to optimizethe program as much as we can to run as fast as it could.

- Find a good enough block size? –**"YES"**
- Use multiple threads? –**"NO"**

Why it's a bad idea to read a file useing multiple threads.

It's a bad idea to file using multiple threads. Threads can get you more CPU cycles if you have enough cores but you still have only one hard disk. So inevitably threads cannot improve the speed of reading file data.

They make it much worse. Reading data from a file is fastest when you access the file sequentially. That minimizes the number of reader's head seeks, by far the most expensive operation on a disk drive. By splitting the reading across multiple threads, each reading a different part of the file, you are making the reader's head constantly jump back and forth. Which will make reading inefficient.