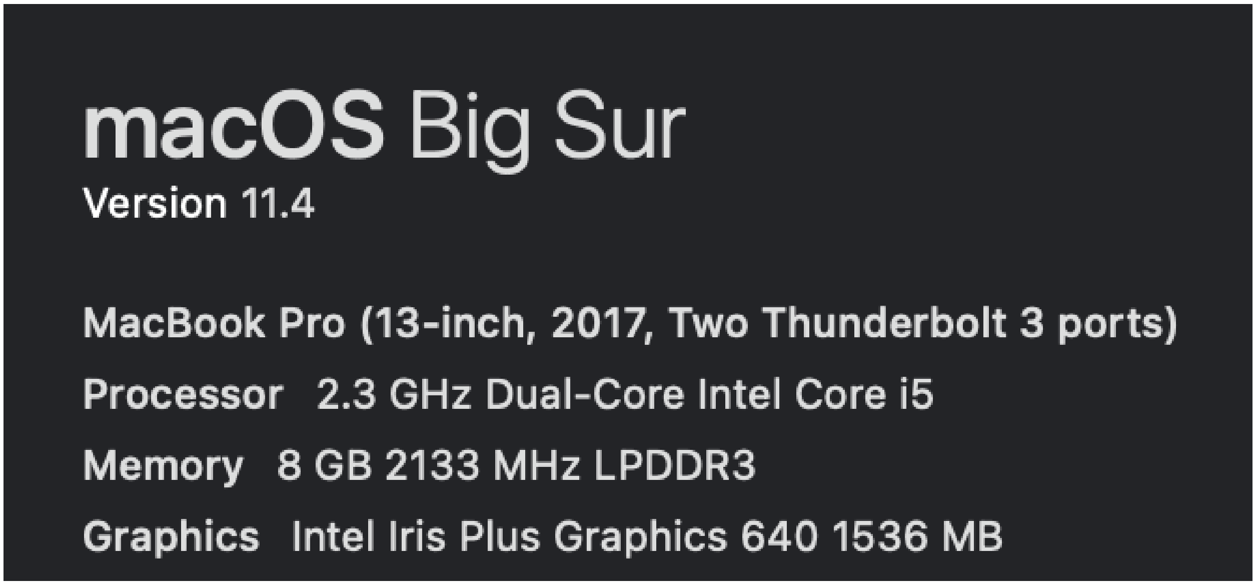
CS-GY 6233 Final Project

Name	Himanshu Kumar	Deepti Gouthaman
Net ID:	hk3427	dg3781
University ID :	N18516342	N12865104
Submission Date	9th December 2021	

1. Hardware Configuration



macOS Big Sur
Version 11.4

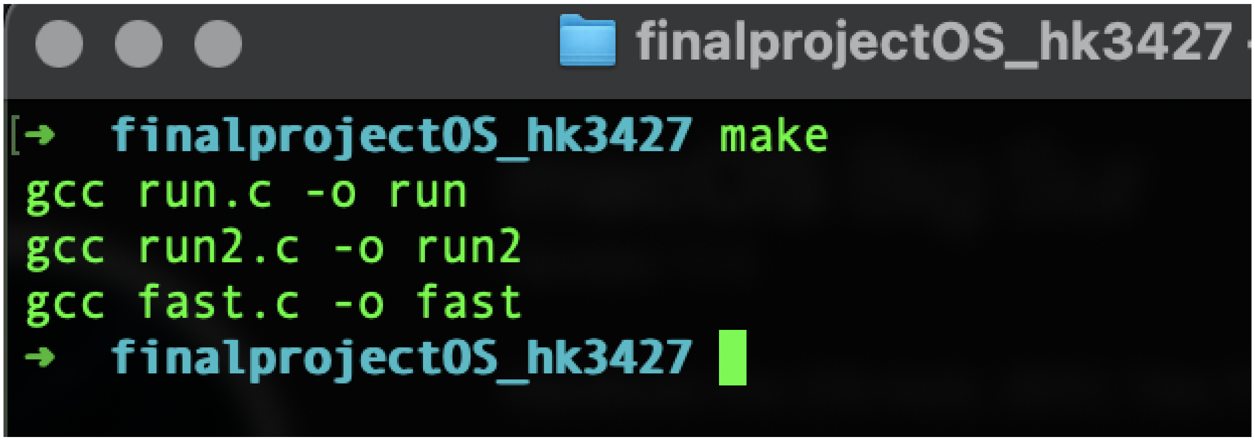
MacBook Pro (13-inch, 2017, Two Thunderbolt 3 ports)
Processor 2.3 GHz Dual-Core Intel Core i5
Memory 8 GB 2133 MHz LPDDR3
Graphics Intel Iris Plus Graphics 640 1536 MB

---

1. Project Instructions

- Download the zip folder named finalprojectOS_hk3427.zip to your local machine.
- Extract the zip.

- Locate the unzipped folder in the terminal.
- Check if these files (run.c, run 2.c, fast.c,Makefile) are present in this directory
- If you wish to use your own file, place the target folder in the same directory. Else, use the write function to create a file of your own size choice.
- If your machine does not have C installed, execute the below command.
    - sudo apt install gcc
- run **make** to generate the executable files.



```
[➜  finalprojectOS_hk3427 make
gcc run.c -o run
gcc run2.c -o run2
gcc fast.c -o fast
➜  finalprojectOS_hk3427
```

- Perform below experiments in the sequence to find the optimal block size to read.

---

2. Basics

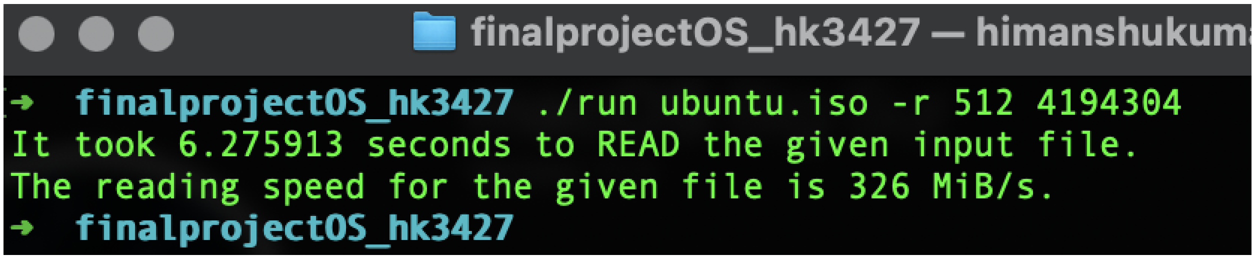
Program to read and write a file using standard C/C++ libraries

Read (Command)

./run <filename> -r blocksize blockcount

Input Example : ./run ubuntu.iso -r 512 4194304

Output :



```
➜  finalprojectOS_hk3427 ./run ubuntu.iso -r 512 4194304
It took 6.275913 seconds to READ the given input file.
The reading speed for the given file is 326 MiB/s.
➜  finalprojectOS_hk3427
```

This command allows the user to input the file name specified by the -r argument along with the blockcount and block size to find the total time it took to read the file. Optionally it also returns the speed performance.

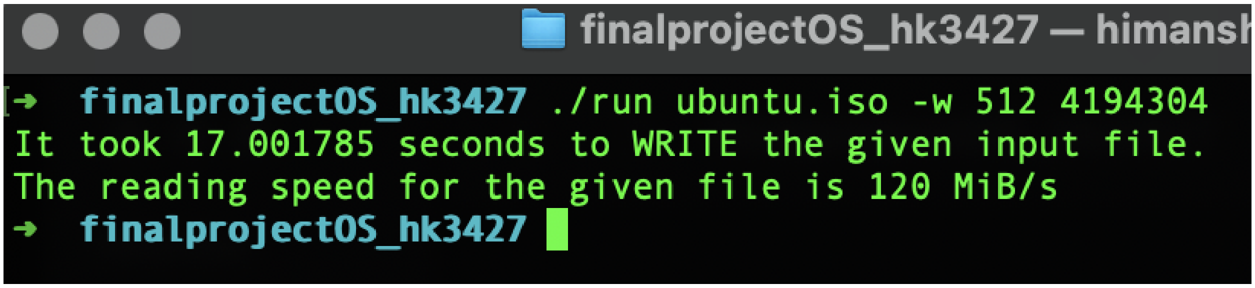
Hint : If you don't know a good blockcount, try running the second experiment first with a given block size to find the blockount for a reasonable read time.

Write (Command)

./run <filename> -w blocksize blockcount

Input Example : ./run <filename> -r blocksize blockcount

Output:



```
[→  finalprojectOS_hk3427 ./run ubuntu.iso -w 512 4194304
It took 17.001785 seconds to WRITE the given input file.
The reading speed for the given file is 120 MiB/s
→  finalprojectOS_hk3427
```

This command allows the user to input the file name specified by the -w argument along with the blockcount and block size to write a file of the desired size. If the file does not exist on your system, it will create a new file with the given name as the second argument on the command line. Optionally it also returns the speed performance.

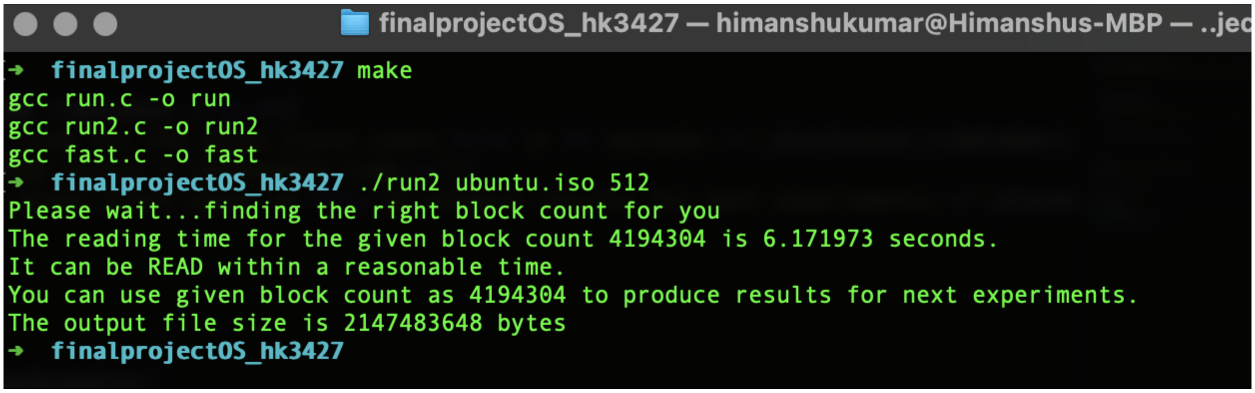
---

3. Measurement

Program to find a file size that can be read within a reasonable time.

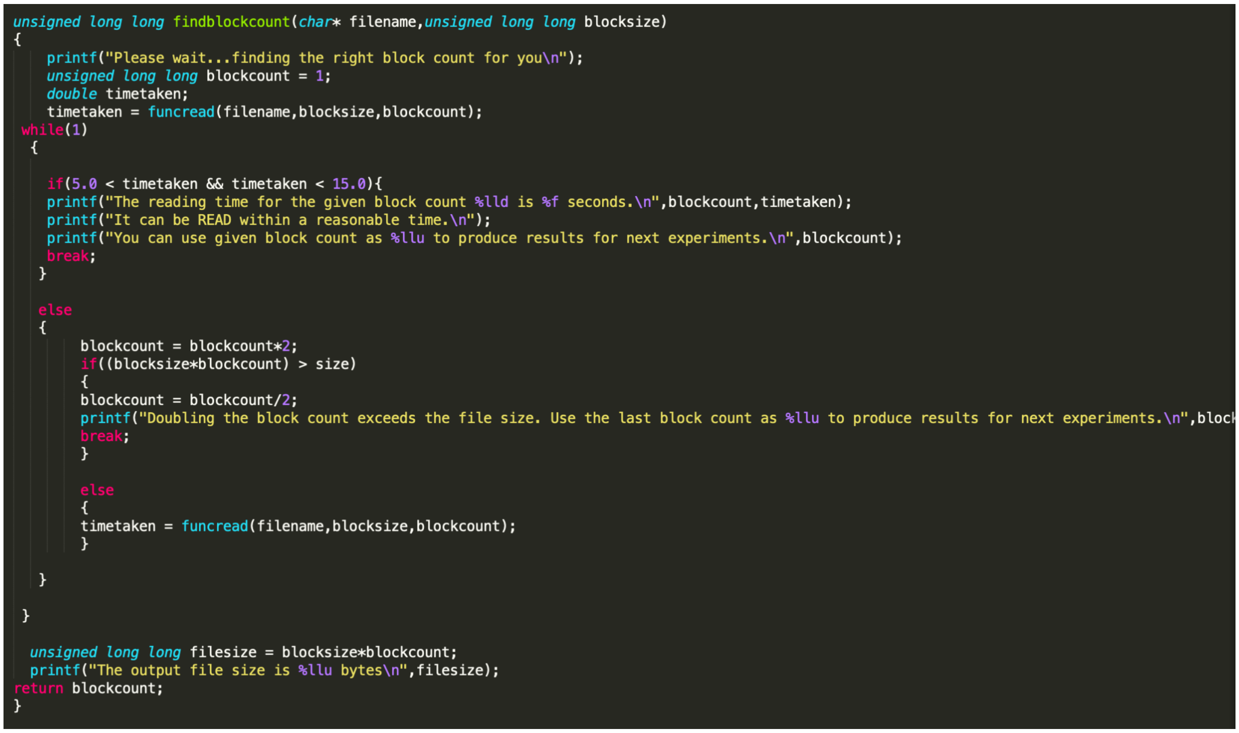
Run Command : ./run2 <filename> <block size>

Input example : ./run2 ubuntu.iso 512



```
[→  finalprojectOS_hk3427 make
gcc run.c -o run
gcc run2.c -o run2
gcc fast.c -o fast
→  finalprojectOS_hk3427 ./run2 ubuntu.iso 512
Please wait...finding the right block count for you
The reading time for the given block count 4194304 is 6.171973 seconds.
It can be READ within a reasonable time.
You can use given block count as 4194304 to produce results for next experiments.
The output file size is 2147483648 bytes
→  finalprojectOS_hk3427
```

This command allows you to find a right block count as displayed above by taking any block size as input. The idea behind this execution is that it starts with a specified block count as 1 and checks the read time by doubling the block count every time. If a reasonable read time is found within the 5 and 15 second, it terminates the loop and returns the blocount found to be fit. Also, internally it checks if the (blockcount*blocksize) does not exceed the user inputted file size.



```
unsigned long long findblockcount(char* filename,unsigned long long blocksize)
{
    printf("Please wait...finding the right block count for you\n");
    unsigned long long blockcount = 1;
    double timetaken;
    timetaken = funcread(filename,blocksize,blockcount);
 while(1)
  {

    if(5.0 < timetaken && timetaken < 15.0){
    printf("The reading time for the given block count %lld is %f seconds.\n",blockcount,timetaken);
    printf("It can be READ within a reasonable time.\n");
    printf("You can use given block count as %llu to produce results for next experiments.\n",blockcount);
    break;
   }

   else
   {
        blockcount = blockcount*2;
        if((blocksize*blockcount) > size)
        {
        blockcount = blockcount/2;
        printf("Doubling the block count exceeds the file size. Use the last block count as %llu to produce results for next experiments.\n",block
        break;
        }

        else
        {
        timetaken = funcread(filename,blocksize,blockcount);
        }

   }

 }

  unsigned long long filesize = blocksize*blockcount;
  printf("The output file size is %llu bytes\n",filesize);
return blockcount;
}
```

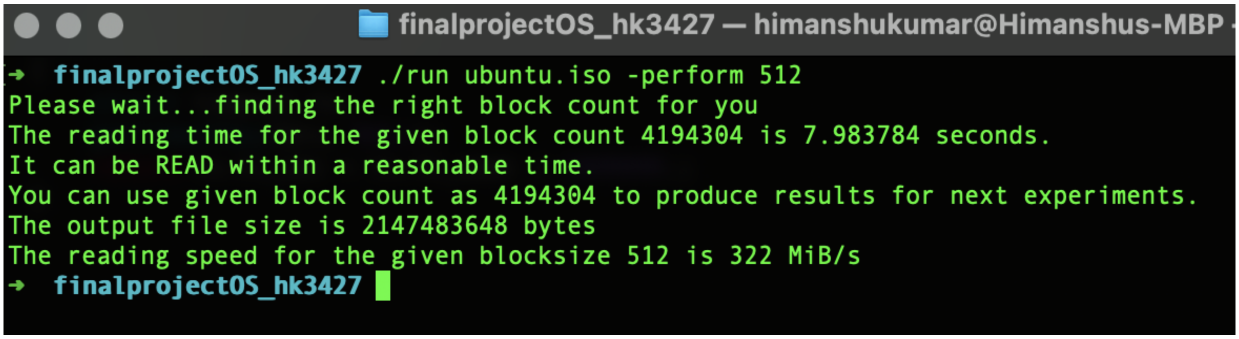
4. Raw Performance / Caching

This program studies the effect of caching on the read system call. While reading the input file, the cache is a part of memory that stores file, shared libraries, data such that future requests can be handled faster. Below mentioned command can be used to test the performance for the given input block size. The program takes a given block size as input and then searches for the desired blockcount. Once a blockcount is found to be read within a reasonable time. It checks the performance for the same.

Run Command : ./run <filename> -perform <block size>

Input example : ./run ubuntu.iso -perform 512

Output :



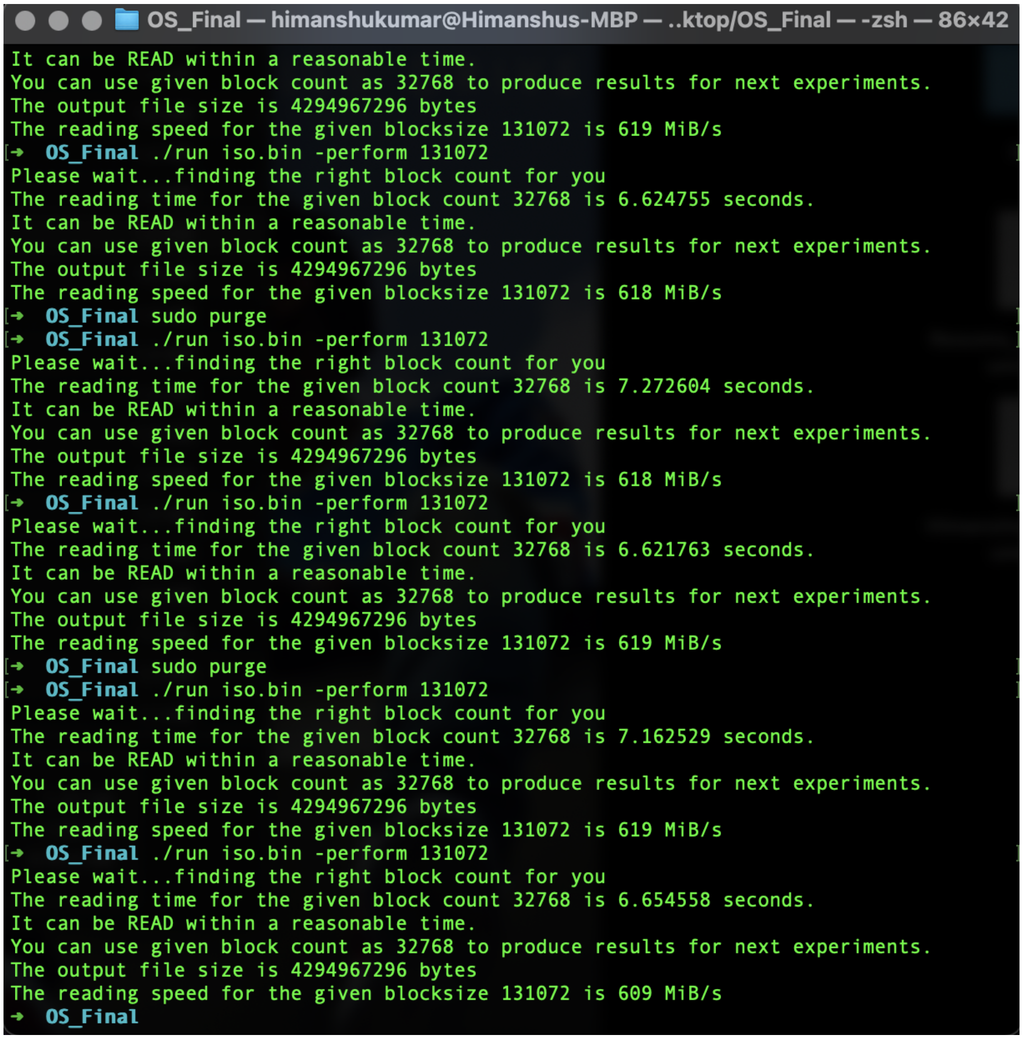
```
→  finalprojectOS_hk3427 ./run ubuntu.iso -perform 512
Please wait...finding the right block count for you
The reading time for the given block count 4194304 is 7.983784 seconds.
It can be READ within a reasonable time.
You can use given block count as 4194304 to produce results for next experiments.
The output file size is 2147483648 bytes
The reading speed for the given blocksize 512 is 322 MiB/s
→  finalprojectOS_hk3427
```

The execution is tested multiple times to find an average read time for the given blocksize. Observations are found as below.

Block Size	**Non-Cache Performance (MiB/s)**				**Cache Performance (MiB/s)**			
	1st Execution	2nd Execution	3rd Execution	4th Execution	1st Execution	2nd Execution	3rd Execution	4th Execution
1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
4	5	5	5	5	4	5	4	5
8	10	10	10	10	10	10	9	10
16	20	21	20	20	20	20	21	20
32	40	40	40	40	40	40	40	40
64	69	74	74	75	74	73	75	75
128	134	134	134	134	132	134	134	134
256	203	218	221	215	221	220	221	221
512	325	327	328	327	329	328	328	328
1024	429	432	432	431	368	432	418	430
2048	501	490	503	502	502	501	501	503
4096	557	551	558	559	560	558	555	559
8192	582	585	586	585	584	585	584	585
16384	601	601	603	599	599	594	594	587

32768	603	564	611	609	604	606	613	612
65536	609	615	600	617	610	617	602	618
131072	619	619	618	619	610	618	610	619
262144	618	620	618	619	618	610	620	610

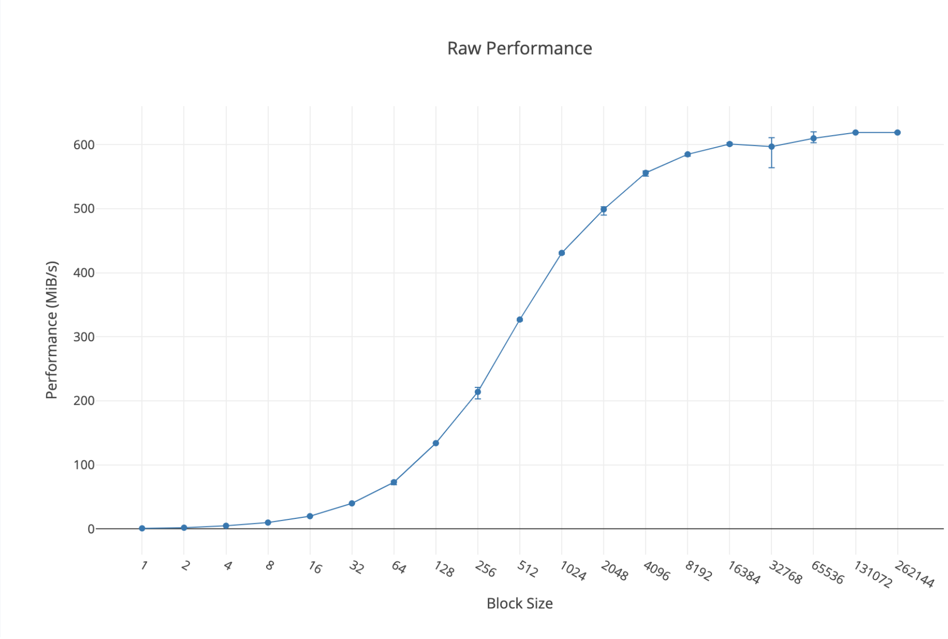
Sample Observation


OS_Final — himanshukumar@Himanshus-MBP — ..ktop/OS_Final — -zsh — 86×42

```
It can be READ within a reasonable time.
You can use given block count as 32768 to produce results for next experiments.
The output file size is 4294967296 bytes
The reading speed for the given blocksize 131072 is 619 MiB/s
[➜  OS_Final ./run iso.bin -perform 131072
Please wait...finding the right block count for you
The reading time for the given block count 32768 is 6.624755 seconds.
It can be READ within a reasonable time.
You can use given block count as 32768 to produce results for next experiments.
The output file size is 4294967296 bytes
The reading speed for the given blocksize 131072 is 618 MiB/s
[➜  OS_Final sudo purge
[➜  OS_Final ./run iso.bin -perform 131072
Please wait...finding the right block count for you
The reading time for the given block count 32768 is 7.272604 seconds.
It can be READ within a reasonable time.
You can use given block count as 32768 to produce results for next experiments.
The output file size is 4294967296 bytes
The reading speed for the given blocksize 131072 is 618 MiB/s
[➜  OS_Final ./run iso.bin -perform 131072
Please wait...finding the right block count for you
The reading time for the given block count 32768 is 6.621763 seconds.
It can be READ within a reasonable time.
You can use given block count as 32768 to produce results for next experiments.
The output file size is 4294967296 bytes
The reading speed for the given blocksize 131072 is 619 MiB/s
[➜  OS_Final sudo purge
[➜  OS_Final ./run iso.bin -perform 131072
Please wait...finding the right block count for you
The reading time for the given block count 32768 is 7.162529 seconds.
It can be READ within a reasonable time.
You can use given block count as 32768 to produce results for next experiments.
The output file size is 4294967296 bytes
The reading speed for the given blocksize 131072 is 619 MiB/s
[➜  OS_Final ./run iso.bin -perform 131072
Please wait...finding the right block count for you
The reading time for the given block count 32768 is 6.654558 seconds.
It can be READ within a reasonable time.
You can use given block count as 32768 to produce results for next experiments.
The output file size is 4294967296 bytes
The reading speed for the given blocksize 131072 is 609 MiB/s
➜  OS_Final
```

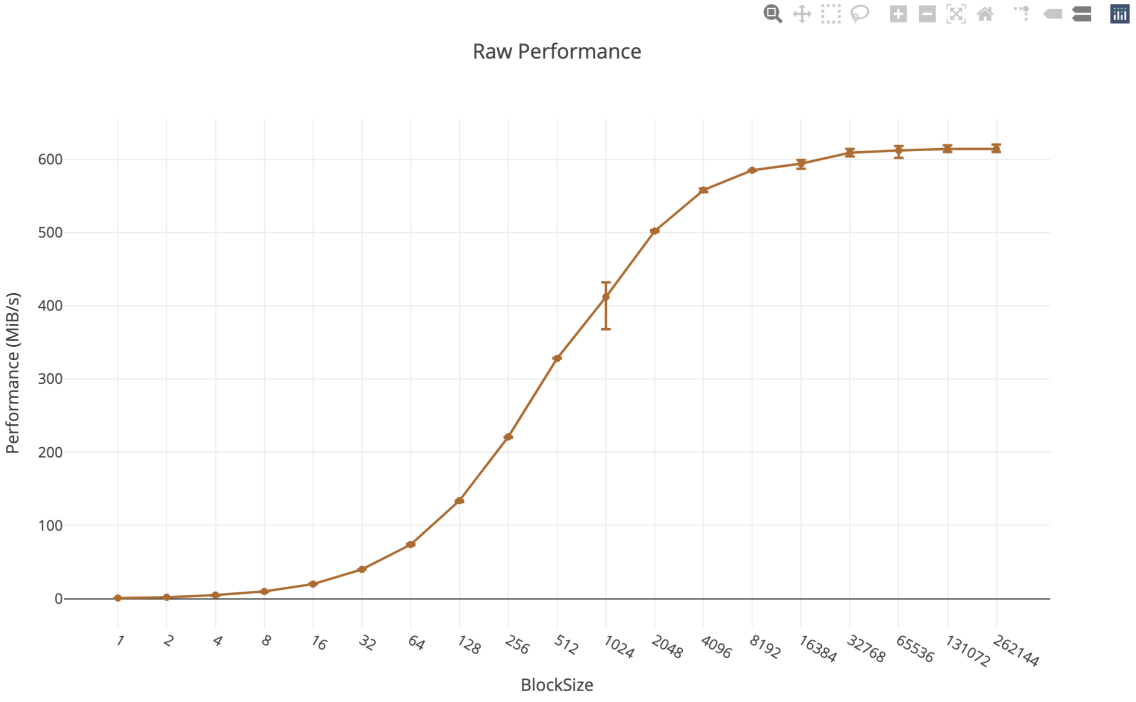
Non-Cached Read Performance

This graph denotes the performance of the read task in MiB/s for different block sizes and it is done by clearing cache, each time before experimenting for the new block size count.



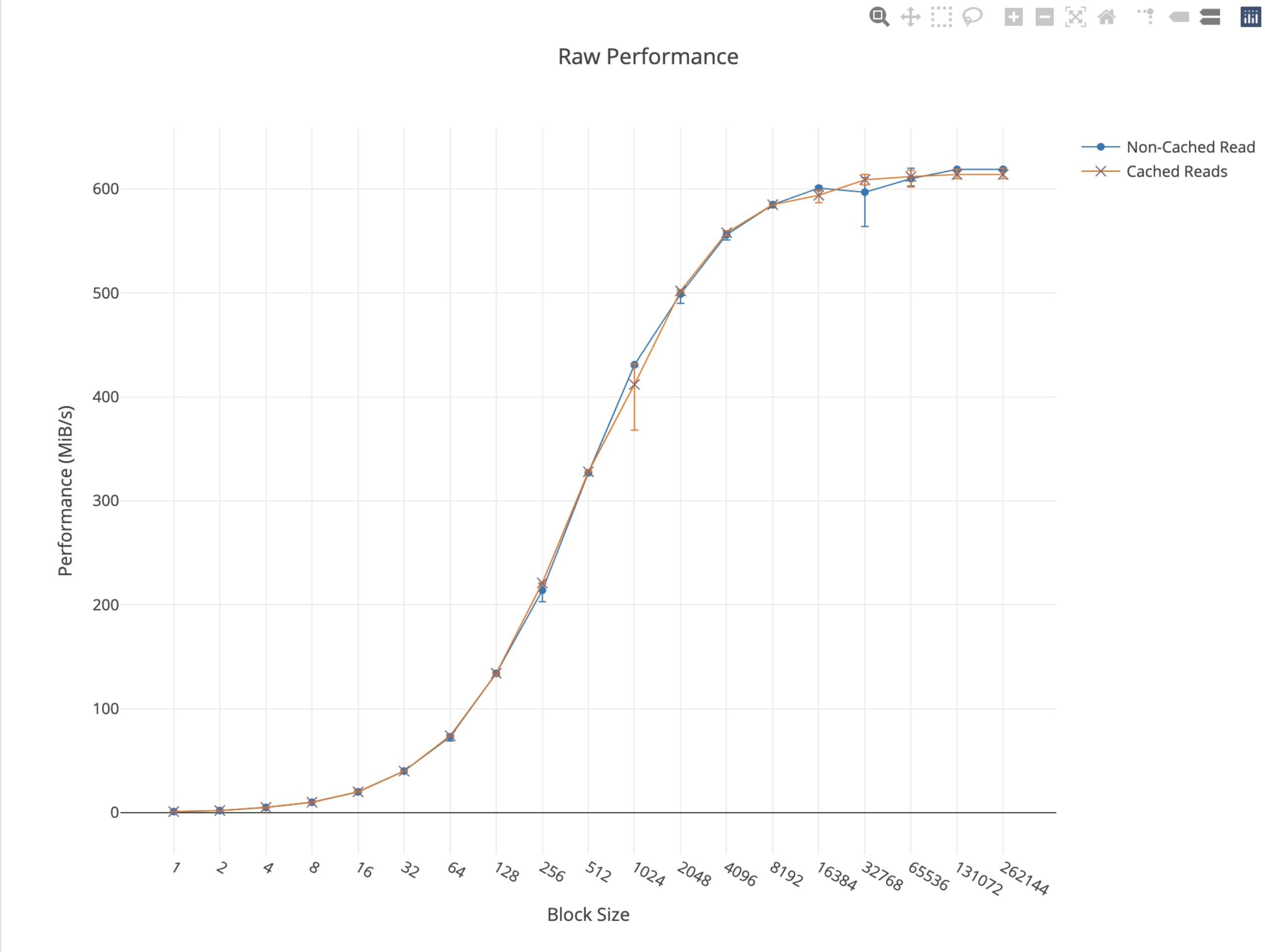
Cached Read Performance

This graph denotes the performance of the read task in MiB/s for different block sizes and the observation is taken immediately after a non-cached read observing the effect of cache.



Non-Cached Read Vs Cached Read Performance

The below graph plots both the **non-cached and cached read** on the same graph and checks for the performance. It is observed that increasing the block count has exponential improvement in the performance speed. Initially for the lower block sizes, the cache does not play a significant role while it does help in the performance while the block size is being increased exponentially as the memory caches the large reads in disk memory while reading.



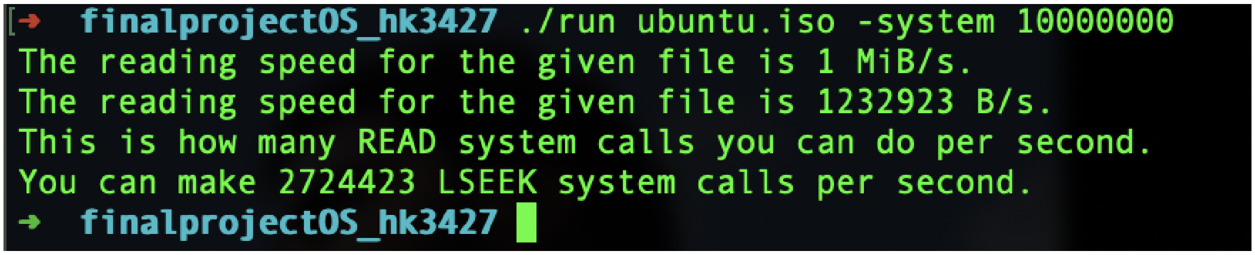
5. System Calls

This experiment is done to study the system calls, which is a programmatic way in which a computer program requests a service from the kernel of the operating system.

Run Command : ./run <filename> -system <block size>

Input example : ./run ubuntu.iso -system 512

Output :



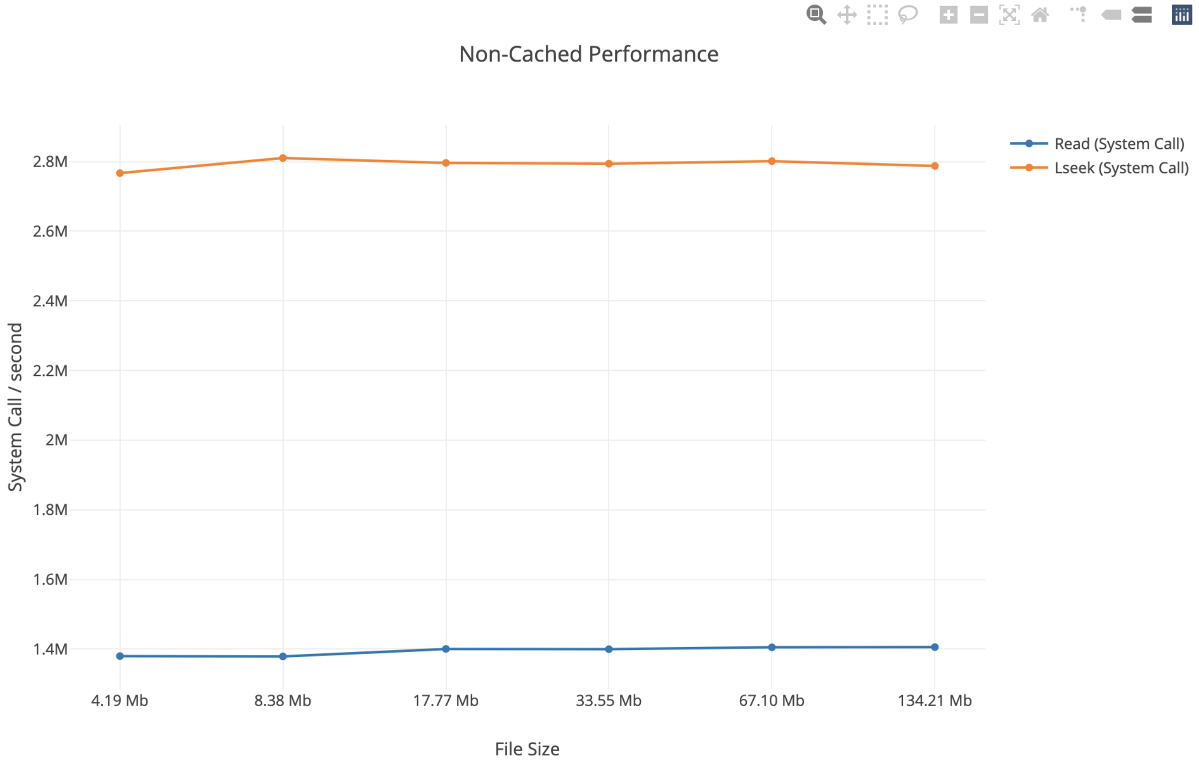
```
[➜  finalprojectOS_hk3427 ./run ubuntu.iso -system 10000000
 The reading speed for the given file is 1 MiB/s.
 The reading speed for the given file is 1232923 B/s.
 This is how many READ system calls you can do per second.
 You can make 2724423 LSEEK system calls per second.
 ➜  finalprojectOS_hk3427 
```

The below mentioned system calls were taken in the consideration for different block count or file sizes.

File Size (MB)	Non-Cached Read Performance (system call/second)	Cached Read Performance (system call/second)	Non Cached Lseek Performance (system call/second)	Cached Lseek Performance (system call/second)	Non-Cached Read Performance (MiB/s)	Cached Read Performance (MiB/s)
4.19 Mb	1379983	1408446	2766851	2810089	1	1
8.38 Mb	1378955	1400119	2810206	2816459	1	1
17.77 Mb	1400519	1410667	2796222	2806853	1	1
33.55 Mb	1399760	1381467	2794054	2572072	1	1

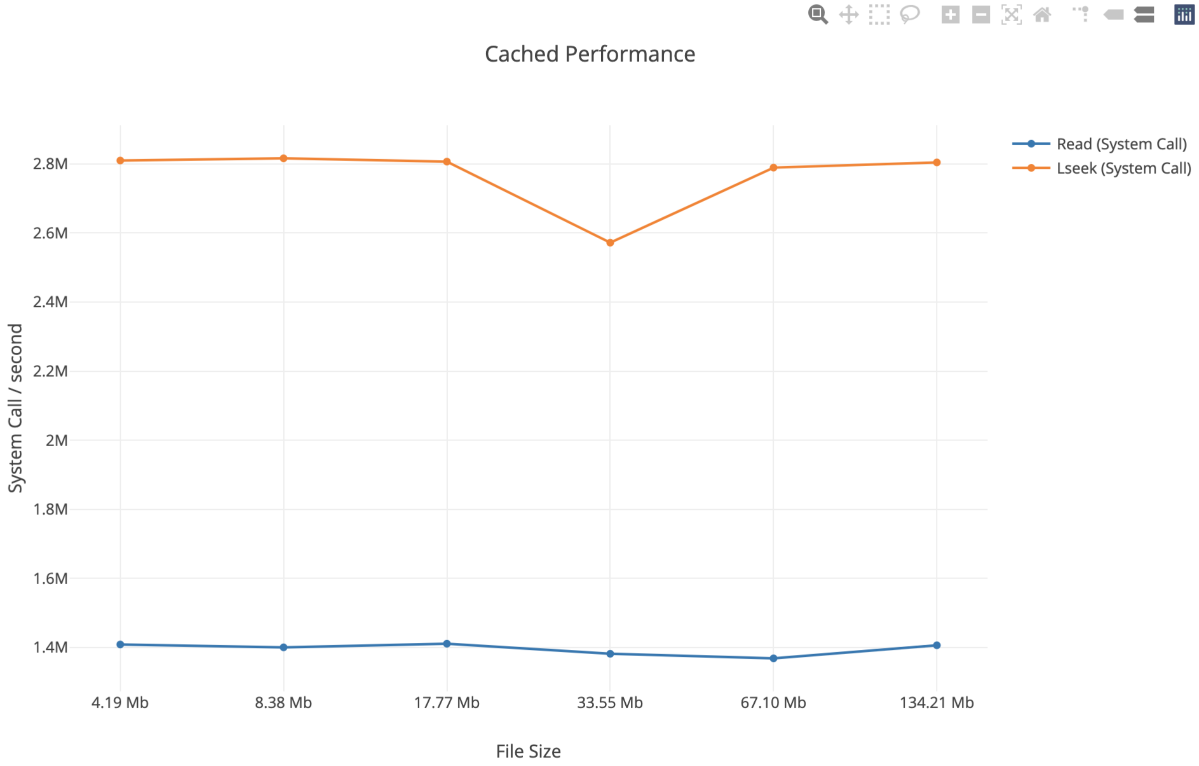
67.10 Mb	1405140	1368233	2801090	2789513	1	1
134.21 Mb	1405797	1406086	2787488	2804583	1	1

System Call (Read,lseek) - NonCached



As we can see from the graph, lseek can nearly perform double the system calls per second in comparison to the read. This is because lseek just returns the offset pointer and does not stress the memory as much as read. Caching plays a very minor role in the change of the observations which can be observed from the below graph.

System Call (Read,lseek) - Cached



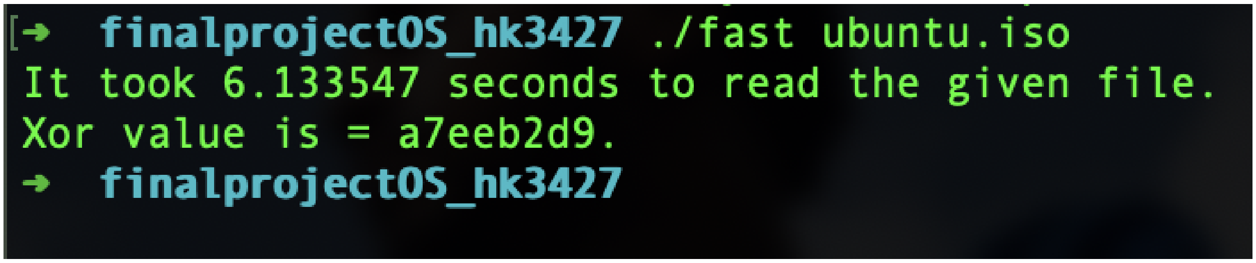
6. Performance (XOR Computation)

This experiment reads the complete file and returns the computed xor value along with the time taken to perform the read of the complete file.

Run Command : ./fast <filename>

Input example : ./fast ubuntu.iso

Output :



```
[➜  finalprojectOS_hk3427 ./fast ubuntu.iso
 It took 6.133547 seconds to read the given file.
 Xor value is = a7eeb2d9.
 ➜  finalprojectOS_hk3427
```