

This is the final project for Intro to OS and it is by its very nature quite open ended.

The environment for this project is real Linux (I recommend a recent Ubuntu, but really any recent distribution should work). You can try to make it work on xv6 for extra credit. You can use C or C++.

I will use discretion when grading it on what deserves [extra] credit. I will provide examples below. If you want to excel at this project you should consider some of the extra credits and possibly come up with more ideas of your own!

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.

Breakdown


1. Basics

- Write a program that can read and write a file from disk using the standard C/C++ library.
 - Add parameter for the file name;
 - Add parameter for how big the file should be (for writing);
 - Add a parameter to specify how much to read with a single call (block size);

2. Measurement

When measuring things it helps if they run for "reasonable" time. It is hard to measure things that run too fast as you need high-precision clocks and a lot of other things can affect the measurement. It is also annoying to wait for a long time for an experiment, especially if you have to do many experiments. For this reason you should make sure that your experiments take "reasonable" time. I recommend something between 5 and 15 seconds.

- Write a program a file size which can be read in "reasonable" time.
 - Input: block size
 - Output: file size

<aside>  Hint: You can start with a small file size and use your program from (1) above to read it and measure the time it takes. You can keep doubling your file size until you get between 5 and 15 seconds.

</aside>

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

Extra credit idea: learn about [Google Benchmark](#) — see if you can use it (note: I have not tried).

3. Raw Performance

Because you would be looking at different file sizes when using different block sizes, it makes sense to use units for performance that are independent of the size. Use MiB/s (this is megabytes per second).

- Make your program output the performance it achieved in MiB/s
- Make a graph that shows its performance as you change the block size.

4. Caching

Once you have a file of "reasonable" size created, reboot your machine.

Call your program to read the file and note the performance.

Call your program to read the file again (immediately after) and note the performance.

Ideally you should observe the second read be much faster, assuming the file can fit in your physical memory. This is the effect of caching.

<aside> 💡 NOTE: On Linux there is a way to clear the disk caches without rebooting your machine. E.g. `sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches"`.

Extra credit: Why "3"? Read up on it and explain.

</aside>

Experiment with clearing the caches and not.

- Make a graph that shows performance for cached and non-cached reads for various block sizes.

5. System Calls

If you have very, very small block size (e.g. 1 byte), most of the time would be spent trapping into the kernel. We talked in class how system calls are expensive. Let's quantify it!

- Measure performance MiB/s when using block size of 1 byte
- Measure performance in B/s. This is how many system calls you can do per second.
- Try with other system calls that arguably do even less real work (e.g. `lseek`)

6. Raw Performance

Try to optimize your program as much as you can to run as fast as it could.

- Find a good enough block size?
- Use multiple threads?
- Report both cached and non-cached performance numbers.

Extra credit idea: Figure out how to do this on high-end hardware. You can try `m5d.metal` instance on AWS cloud. This normally costs money (\$5/hour) but if you use spot pricing you can get it as cheap as \$1/hour. Also it is charged by the minute, so you can bring it up, run your experiment, shut it down for cents on the dollar. Also, Amazon has a program that gives free credits to students... Not sure what their turnaround time is, but you can try!

Make sure you use the NVMe drive which is attached to it.