Project Members:

Andrew Branum - 5291638
Nathaniel Cecil - 5283895

Project Implementation:

The project was implemented fully from the project description.

Source Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <dirent.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#include <sys/stat.h>

#define BYTE 4096
#define LEN 100

int func(const void *p, const void *q){
   return (*(long int*)p - *(long int*)q);
}

float print(long int arr[], int n)
{
   float med = 0;
   if(n%2 == 0){
   med = (arr[(n-1)/2] + arr[n/2])/2;
   }
   else{
   med = arr[n/2];
   }

   return med;
}

int main(int argc, char *argv[]) {
```

```
   int K = atoi(argv[1]);
   //char file_name = (char *) malloc(LEN);
   if(K == 1) { // file creation mode
       char* file_name = argv[2];
       printf("file name: %s\n", file_name);
       size_t file_size = atoi(argv[3]) * BYTE;
       printf("file size: %ld\n", file_size);
       //int creat(&file_name, mode_t mode);
       int file_des = open(file_name, O_CREAT | O_RDWR | O_TRUNC, S_IRWXG
| S_IRWXU | S_IROTH);//O_CREAT = make new file if there isn't one, O_RDWR
= read and write, O_TRUNC = truncate to 0 and set length to 0 if the file
already exists, S_... are all permissions.

       char *buff = (char *) malloc(file_size); // sets buffer the size of
the file
       memset(buff, 0, file_size); //fills buffer with 0s
       printf("buff 1 & 2: %d %d\n", buff[0], buff[1]);
       ssize_t f_write = write(file_des,buff, file_size);//writes buffer
to file
       if(file_des == -1){
       printf("file_des Error #: %d\n", file_des);

       }
       if(f_write == -1){
       printf("f_write Error #: %ld\n", f_write);

       }
       int f = fsync(file_des);
       if(f == -1){
       printf("fsync Error #: %d\n", errno);

       }
       free(buff);
   }

   if(K == 2){
       struct timespec time;
       struct stat buf;
```

```
        char* file_name = argv[2];
        size_t IO_size = atoi(argv[3]);
        int samples = atoi(argv[4]);
        long int tim[samples];


        int file_des = open(file_name, O_RDWR);
        stat(file_name, &buf);
        int num = buf.st_size/BYTE;

    int i = 0;
        while(i != samples){
        int rand = random() % num;
        off_t off = rand * BYTE;

        while(off + IO_size  > buf.st_size){
            off -= 4096;
        }

        clock_gettime(CLOCK_MONOTONIC, &time);
        time_t curtime = time.tv_nsec * 1000; //microseconds

        //int pos = lseek(file_des, off, SEEK_SET);
        char* buffer [IO_size];
        read(file_des, buffer, IO_size);

        clock_gettime(CLOCK_MONOTONIC, &time);
        time_t endtime = time.tv_nsec * 1000; //microseconds
        time_t total_time = endtime - curtime;
        tim[i] = total_time;
        i++;
        }

        qsort(tim, samples, sizeof(long int), func);
        float med = print(tim, samples);
        med = med / 100000;
        printf("IO Latency Median: %0.2f\n", med);
    }
    return 0;
}
```

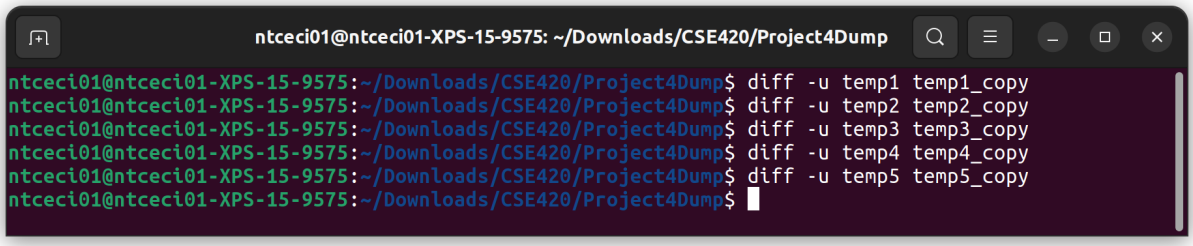
Experimental Results:

Experiment 1: File Creation

```
// Calling ./myio
sudo ./myio 1 temp1 100 - 0m0.017s
sudo ./myio 1 temp2 1000 - 0m0.037s
sudo ./myio 1 temp3 10000 - 0m0.139s
sudo ./myio 1 temp4 100000 - 0m0.941s
sudo ./myio 1 temp5 500000 - 0m3.370s

// Using dd
dd if=temp1 of=temp1_copy
800+0 records in
800+0 records out
409600 bytes (410 kB, 400 KiB) copied, 0.0168819 s, 24.3 MB/s

dd if=temp2 of=temp2_copy
8000+0 records in
8000+0 records out
4096000 bytes (4.1 MB, 3.9 MiB) copied, 0.121842 s, 33.6 MB/s

dd if=temp3 of=temp3_copy
80000+0 records in
80000+0 records out
40960000 bytes (41 MB, 39 MiB) copied, 0.46107 s, 88.8 MB/s

dd if=temp4 of=temp4_copy
800000+0 records in
800000+0 records out
409600000 bytes (410 MB, 391 MiB) copied, 3.72905 s, 110 MB/s

dd if=temp5 of=temp5_copy
4000000+0 records in
4000000+0 records out
2048000000 bytes (2.0 GB, 1.9 GiB) copied, 17.9298 s, 114 MB/s
```

As displayed by the execution time results above, the ‘dd’ command is significantly slower than the ‘myio’ program.

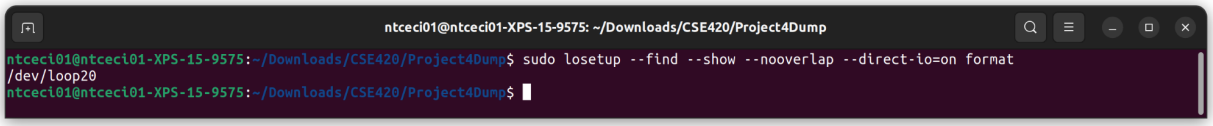
Using the ‘diff’ command shows no differences between the two files



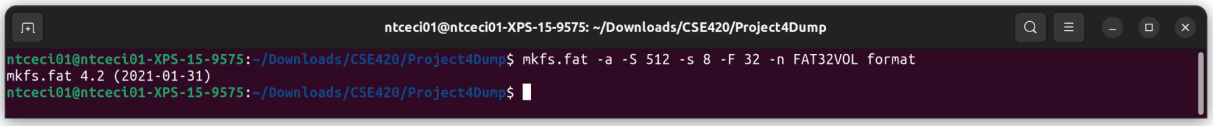
Experiment 2: Formatting and Mounting FAT32

using -> sudo ./myio 1 format 500000 to create a 2.0GB large file

loop device:

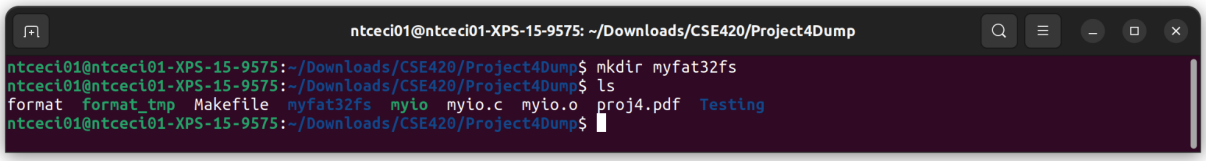


format virtual disk:

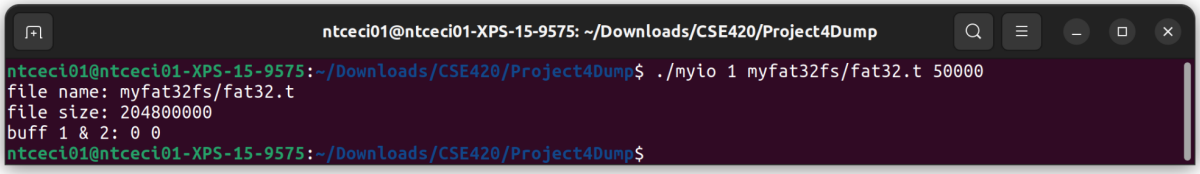


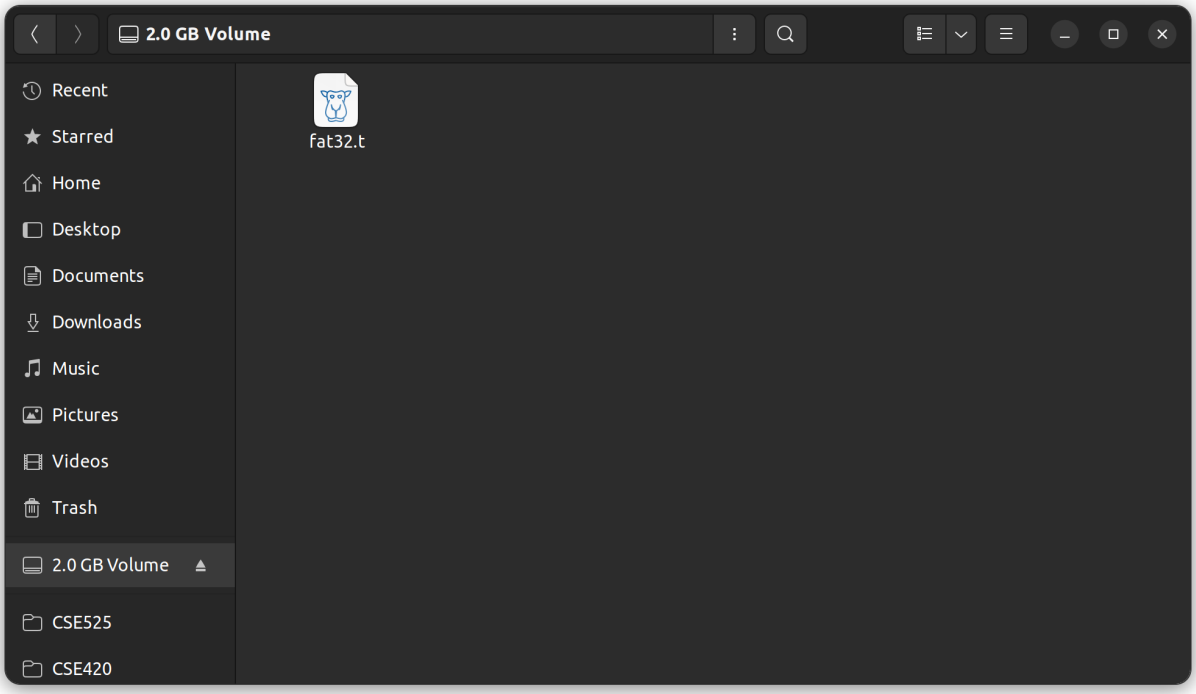
-a: disables alignment to provide a handful of additional clusters of storage
-S 512: specify the number of bytes per logical sector. Must be a power of 2 >= 512
-s 8: specifies the number of disk sectors per cluster. Must be a power of 2
-F 32: FAT-SIZE, specifies the type of file allocation tables used
-n FAT32VOL: VOLUME-NAME, sets the volume name of the filesystem

make directory:



mount volume:

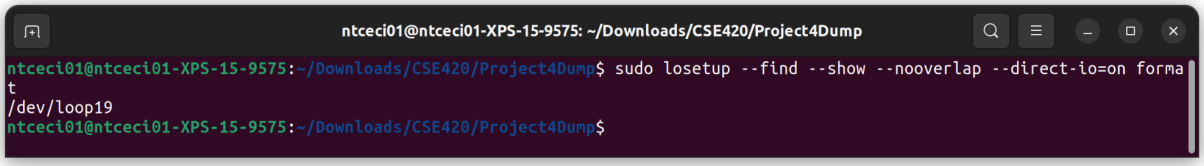




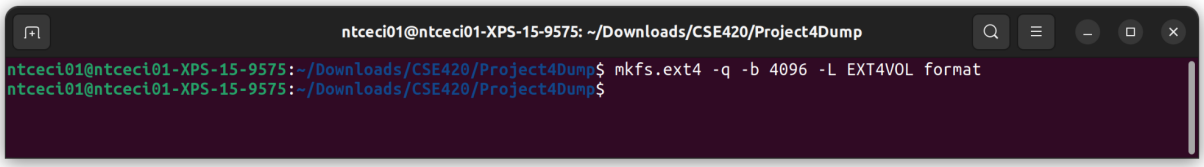
Experiment 3: Formatting and Mounting ext4

using -> sudo ./myio 1 format 500000 to create a 2.0GB large file

loop device:



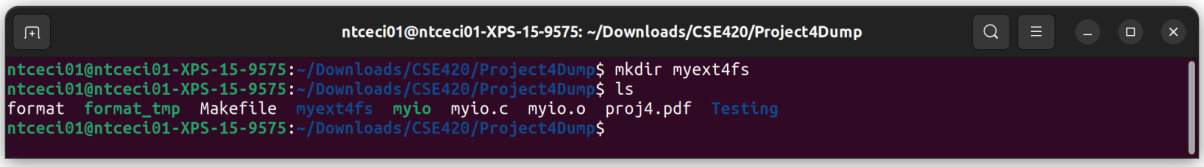
format virtual disk:



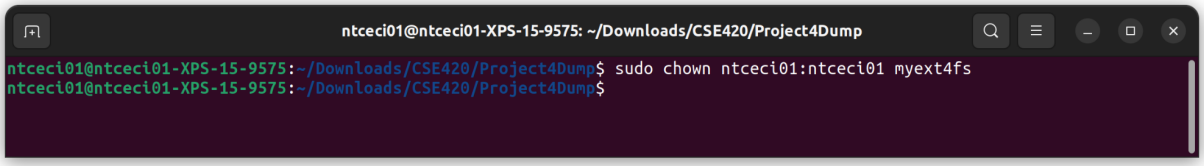
-q: quiet execution, useful if being running in a script (like test.sh)
-b: block-size, specifies the size of blocks in bytes
-L: new-volume-label, sets the volume label of the file system

All of these parameters of different from the FAT32 mounting, although -L performs the same function of -n from FAT32

make directory:

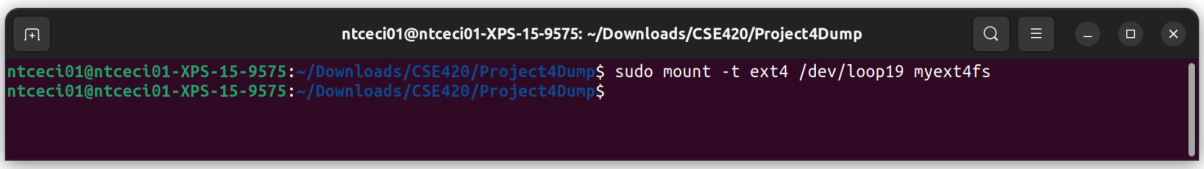


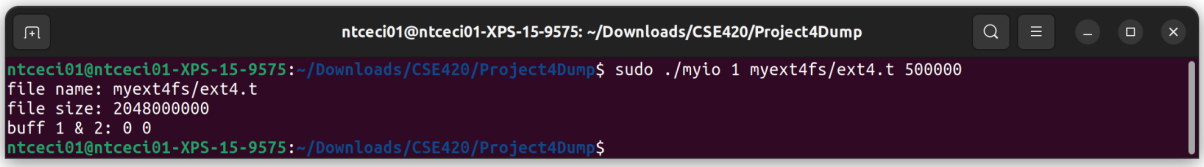
chown:

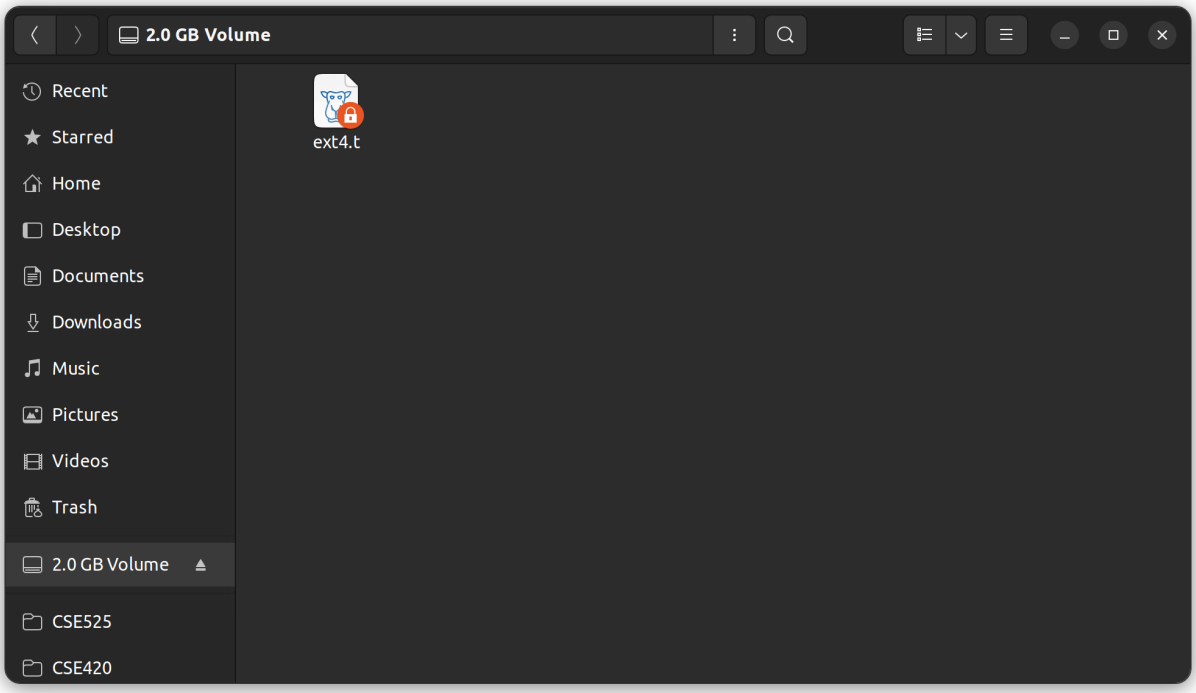


This is different than before. It stands for change file owner and group. This is used to manage ownership and permissions of are secure by putting restrictions on who can modify files depending on if different users are accessing the operating system.

mount volume:

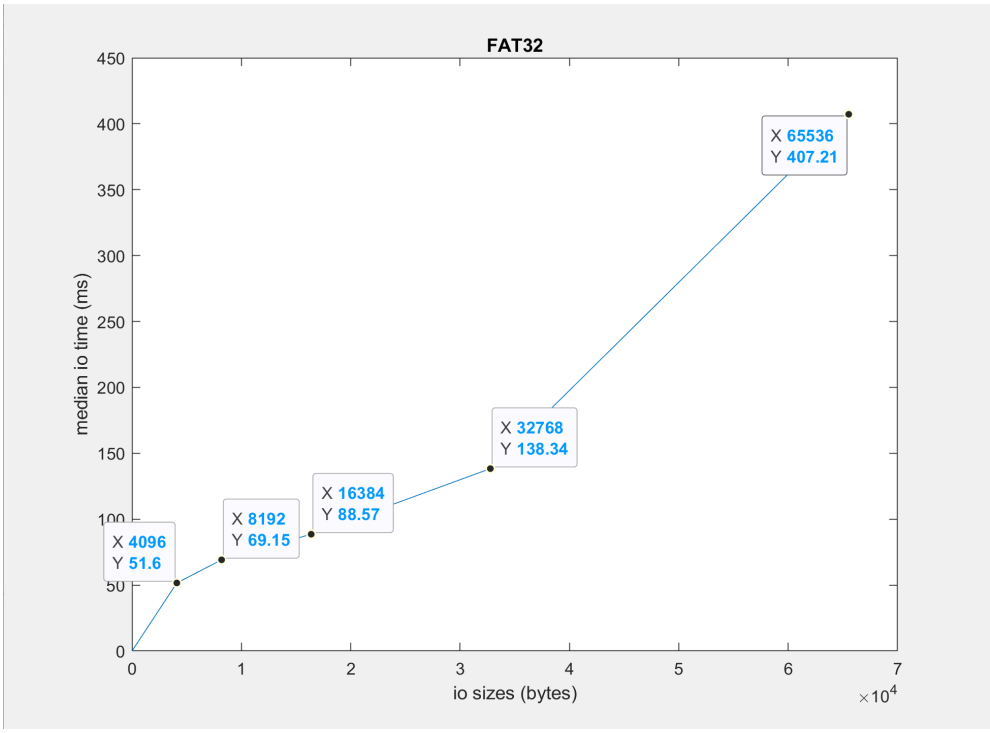






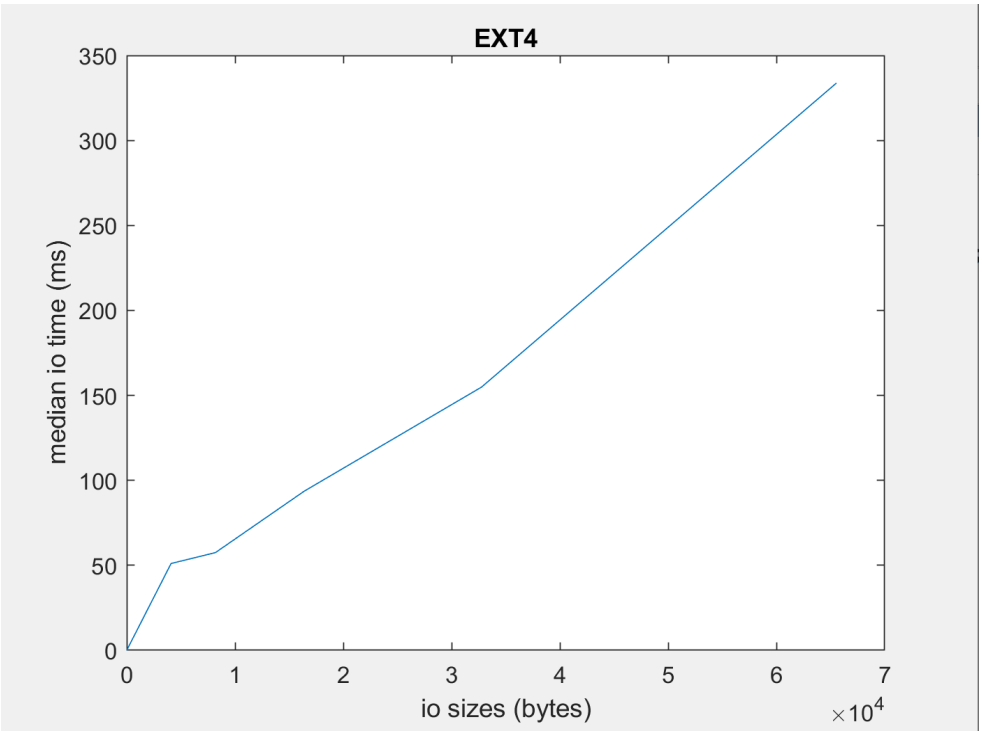
Experiment 4: Random IO Experiments

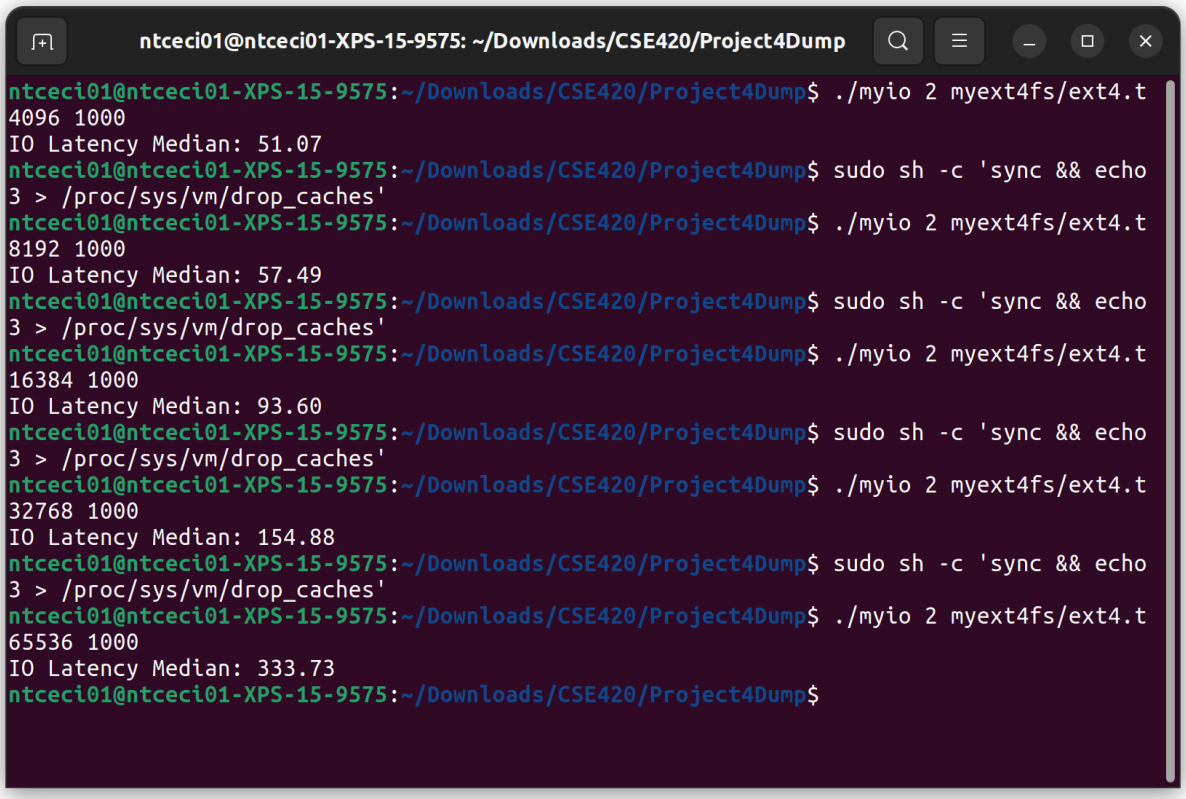
FAT32 plot:



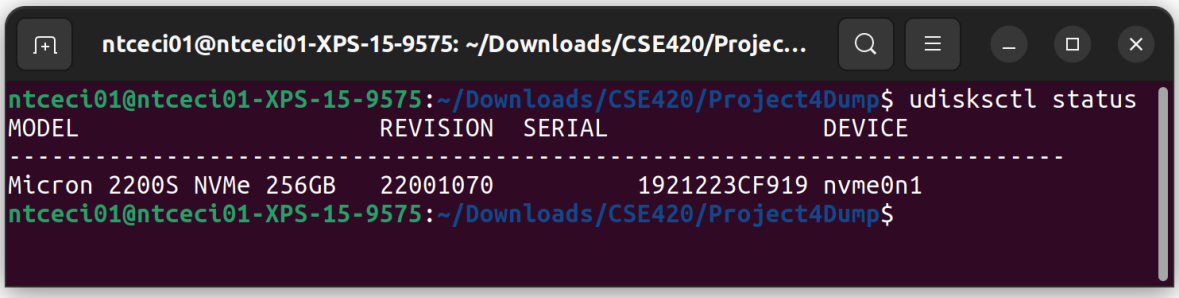
```
ntceci01@ntceci01-XPS-15-9575: ~/Downloads/CSE420/Project4Dump
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ ./myio 2 myfat32fs/fat32.t
4096 1000
IO Latency Median: 51.60
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ sudo sh -c 'sync && echo 3
> /proc/sys/vm/drop_caches'
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ ./myio 2 myfat32fs/fat32.t
8192 1000
IO Latency Median: 69.15
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ sudo sh -c 'sync && echo 3
> /proc/sys/vm/drop_caches'
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ ./myio 2 myfat32fs/fat32.t
16384 1000
IO Latency Median: 88.57
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ sudo sh -c 'sync && echo 3
> /proc/sys/vm/drop_caches'
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ ./myio 2 myfat32fs/fat32.t
32768 1000
IO Latency Median: 138.34
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ sudo sh -c 'sync && echo 3
> /proc/sys/vm/drop_caches'
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ ./myio 2 myfat32fs/fat32.t
65536 1000
IO Latency Median: 407.21
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$ sudo sh -c 'sync && echo 3
> /proc/sys/vm/drop_caches'
ntceci01@ntceci01-XPS-15-9575:~/Downloads/CSE420/Project4Dump$
```

ext4 plot:





storage device specs:



These results make sense, they scale at roughly the same rate as io_size increases.

Experiment 5: Cleanup