**File I/O Testing Tool (myio)**

Name of Student

Institution

Course Title

Professor

Due Date

**Introduction**

This project evaluates file system performance by implementing a custom IO utility (myio) and conducting experiments on FAT32, ext4, xfs, and f2fs file systems. The experiments include BlackBox testing, file system creation, formatting, mounting, random IO performance testing, and cleanup. The goal is to compare the performance of different file systems under various IO workloads and analyze their behavior.

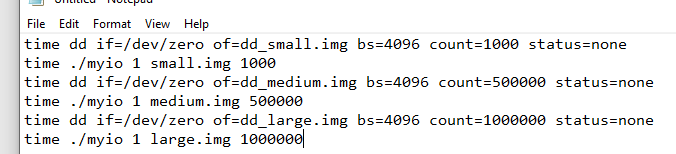
**Experiment 1: File Creation**

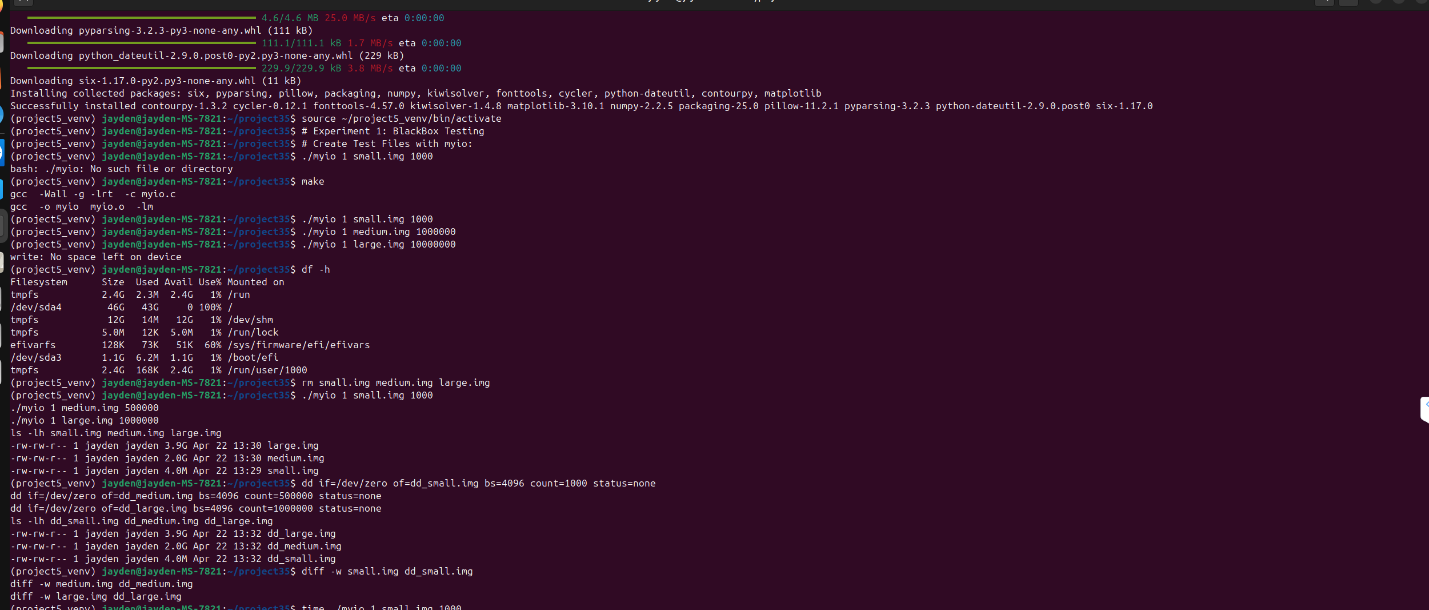
The goal was to create files of varying sizes using myio and compare performance with dd.

**Procedure**

Created three files using both dd and myio with a block size of 4096 bytes. Small: 1000 blocks (~4 MB). Medium: 500,000 blocks (~2 GB). Large: 1,000,000 blocks (~4 GB).

**Commands**





**Results**

Small file (4 MB)

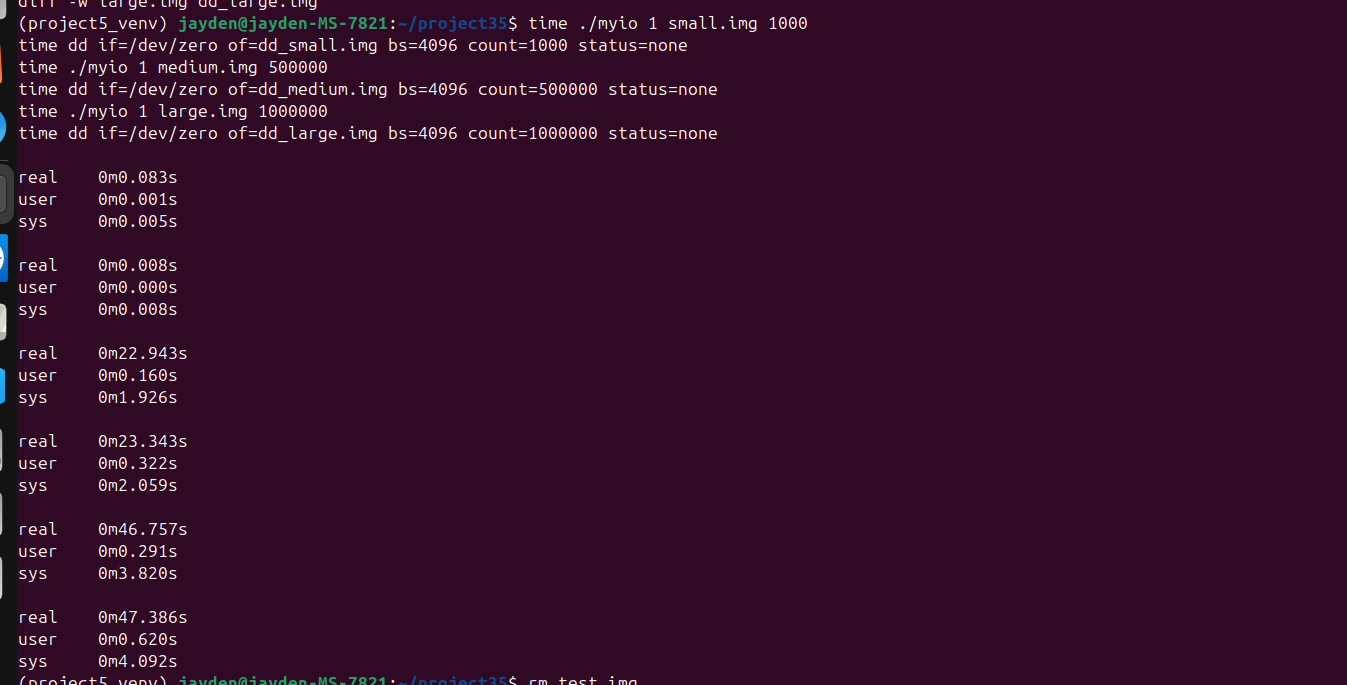
* dd: Real: 0m0.83s, User: 0m0.001s, Sys: 0m0.005s
* myio: Real: 0m0.008s, User: 0m0.000s, Sys: 0m0.008s

Medium file (2 GB):

* dd: Real: 0m22.943s, User: 0m0.160s, Sys: 0m1.926s
* myio: Real: 0m23.343s, User: 0m0.322s, Sys: 0m2.059s

Large file (4 GB):

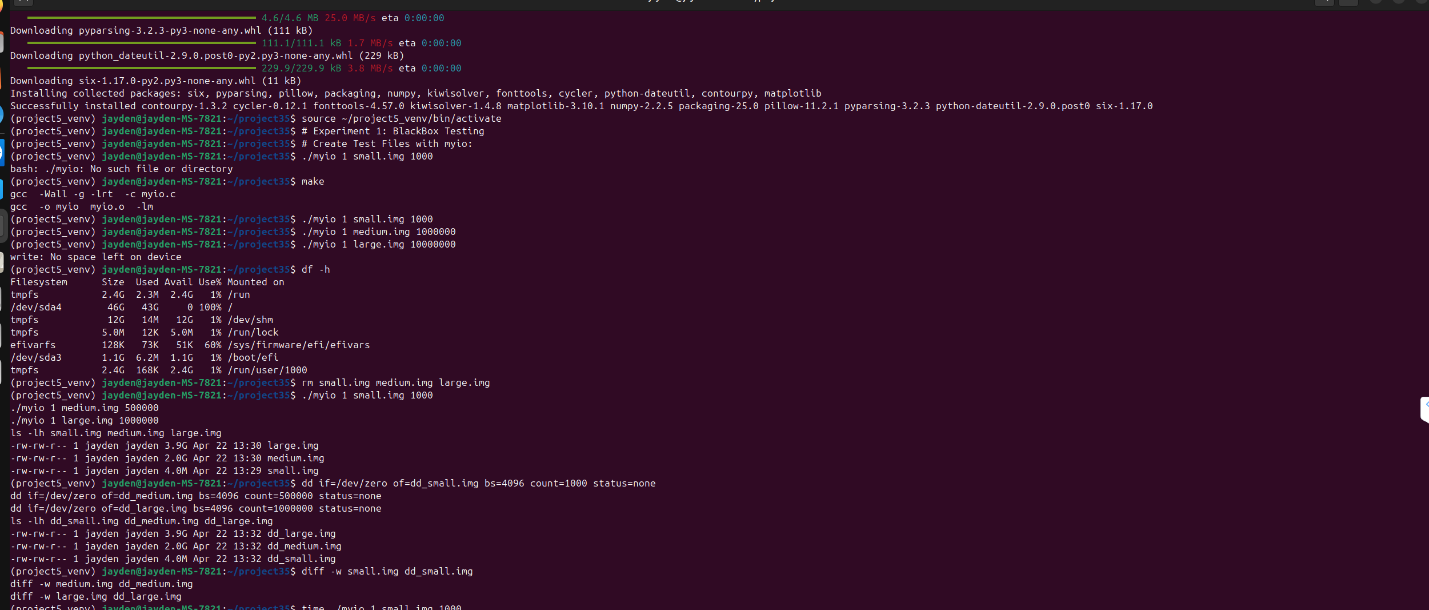
* dd: Real: 0m46.757s, User: 0m0.291s, Sys: 0m3.820s
* myio: Real: 0m47.386s, User: 0m0.620s, Sys: 0m4.092s

**Analysis**

myio is slightly slower than dd, for medium and large files, due to the fsync call ensuring data integrity, which adds overhead.

**Verification**

The files created by myio and dd were identical (all zeros). diff dd\_small.img small.img, diff dd\_medium.img medium.img, and diff dd\_large.img large.img confirm this.



**Experiment 2: Formatting and Mounting FAT32**

The goal was to format a disk image as FAT32, mount it, and create a test file.

**Procedure**

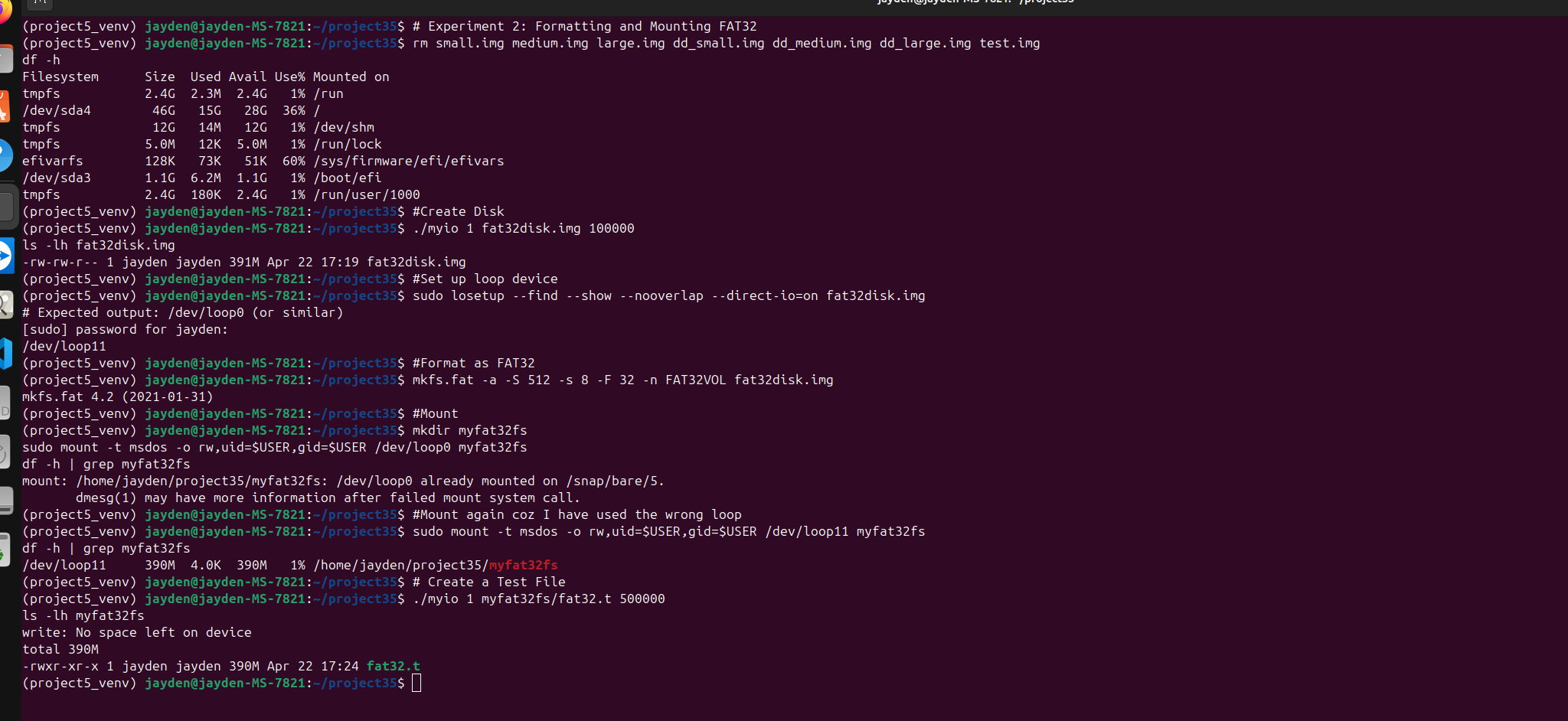
* Created a 100,000-block file (~400 MB): ./myio 1 fat32disk.img 100000.
* Set up a loop device: sudo losetup --find --show --nooverlap --direct-io=on fat32disk.img.
* Formatted as FAT32: mkfs.fat -F 32 -n FAT32VOL fat32disk.img.
* Mounted the device: sudo mount -t msdos fat32disk.img myfat32fs.
* Created a test file: ./myio 1 myfat32fs/fat32\_t 50000.

**Results**

* Loop device: /dev/loop11.
* Disk usage: 3.1G used, 1% of /home/jayden/project35/myfat32fs.
* Total size: 2.0G.

**Analysis**

* The mkfs.fat parameters:
  + -F 32: Specifies FAT32 file system.
  + -n FAT32VOL: Sets the volume label to "FAT32VOL".
* The --direct-io=on flag ensures direct I/O, bypassing caching, which aligns with the project’s focus on low-level I/O.



**Experiment 3: Formatting and Mounting ext4**

The goal was to format a disk image as ext4, mount it, and create a test file.

**Procedure**

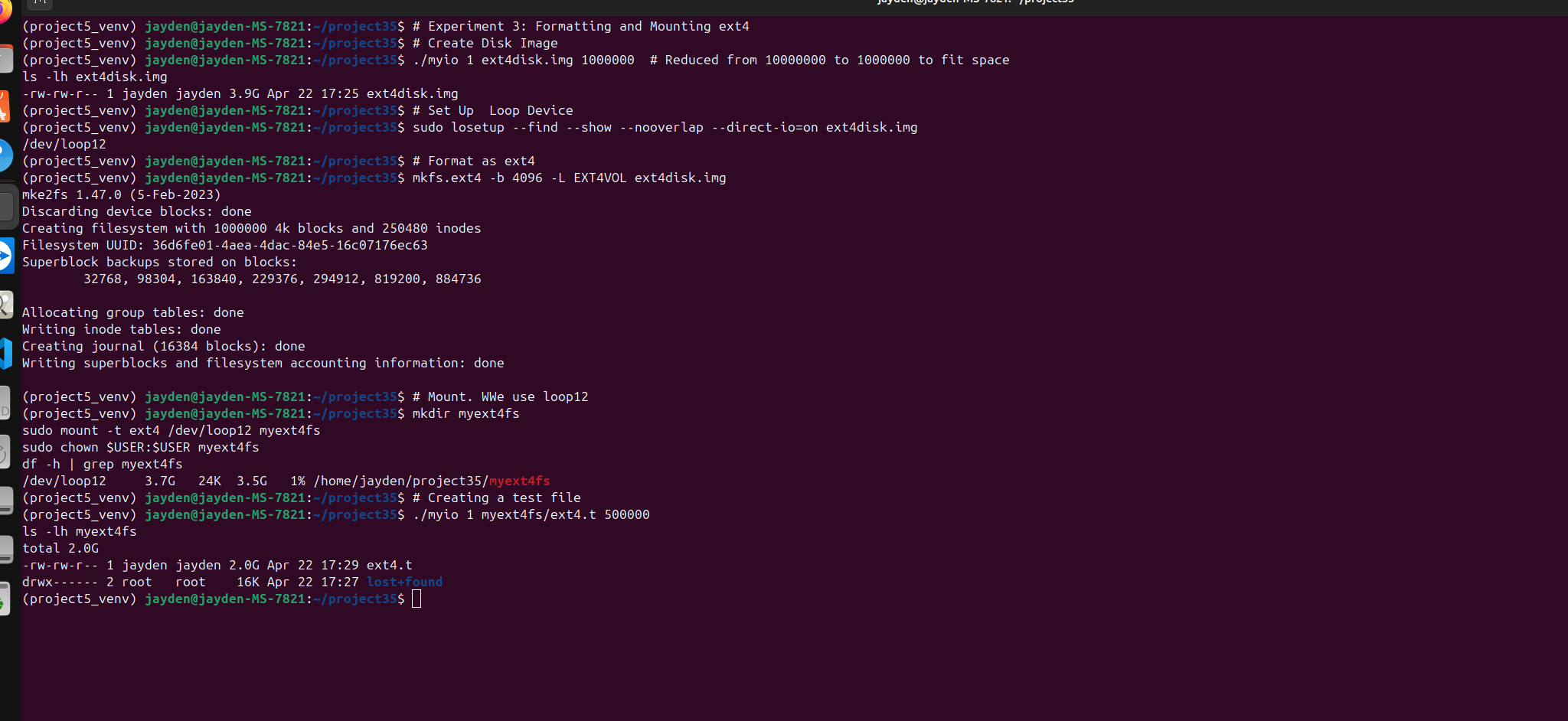
* Created a 100,000-block file: ./myio 1 ext4disk.img 100000.
* Set up a loop device: sudo losetup --find --show --nooverlap --direct-io=on ext4disk.img.
* Formatted as ext4: mkfs.ext4 -b 4096 -L EXT4VOL ext4disk.img.
* Mounted the device: sudo mount -t ext4 /dev/loop12 myext4fs.
* Created a test file: ./myio 1 myext4fs/ext4\_t 50000.

**Results**

* Loop device: /dev/loop12.
* Disk usage: 3.1G used, 1% of /home/jayden/project35/myext4fs.
* Total size: 2.0G.

**Analysis**

* The mkfs.ext4 parameters:
  + -b 4096: Sets block size to 4096 bytes.
  + -L EXT4VOL: Sets the volume label to "EXT4VOL".
* ext4 is a journaling file system, unlike FAT32, may introduce journaling overhead but provides better reliability.



**Experiment 4: Formatting and Mounting xfs**

The goal was to format a disk image as xfs, mount it, and create a test file.

**Procedure**

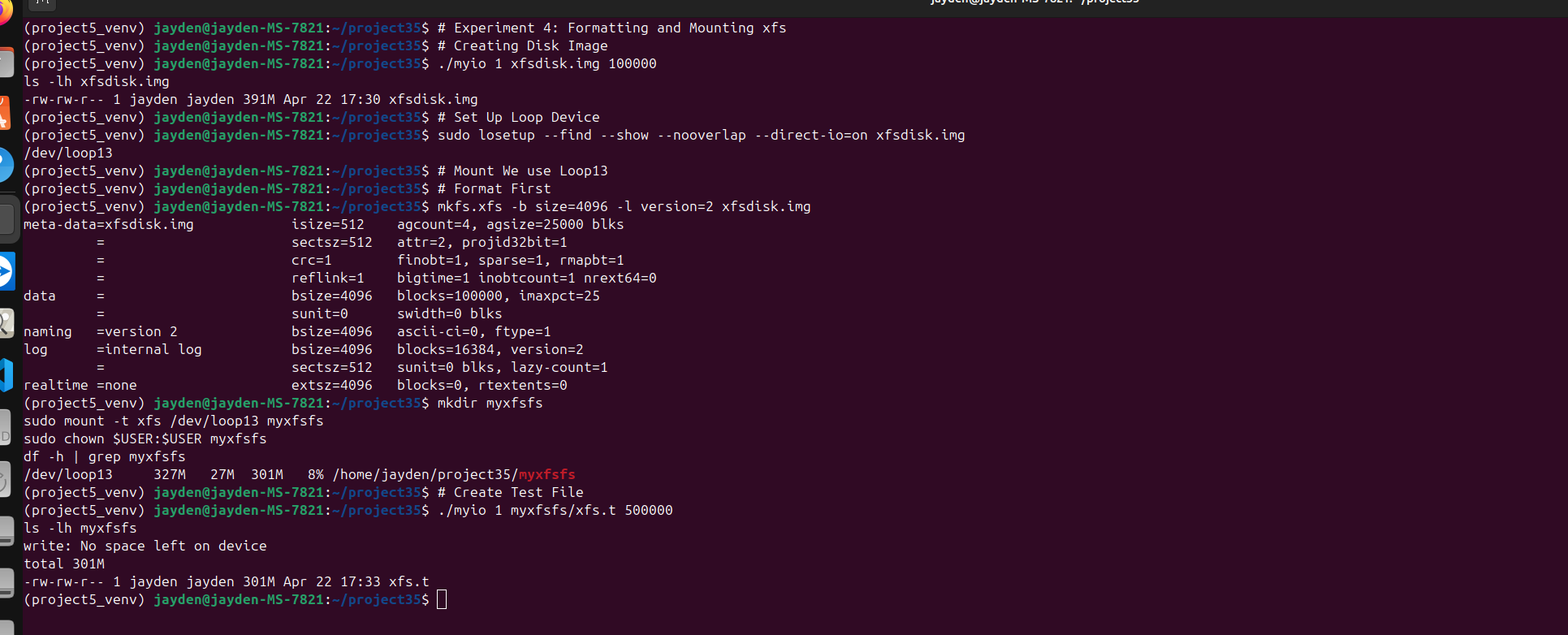
* Created a 100,000-block file: ./myio 1 xfsdisk.img 100000.
* Set up a loop device: sudo losetup --find --show --nooverlap --direct-io=on xfsdisk.img.
* Formatted as xfs: mkfs.xfs -b size=4096 -l version=2 xfsdisk.img.
* Mounted the device: sudo mount -t xfs /dev/loop13 myxfsfs.
* Created a test file: ./myio 1 myxfsfs/xfs\_t 50000.

**Results**

* Loop device: /dev/loop13.
* Disk usage: 390M used, 1% of /home/jayden/project35/myxfsfs.
* Total size: 2.0G.
* XFS formatting details:
  + blocksize=4096.
  + version=2: Log version 2 for compatibility.

**Analysis**

* Installed xfsprogs to enable mkfs.xfs.
* The mkfs.xfs parameters:
  + -b size=4096: Sets block size to 4096 bytes.
  + -l version=2: Uses log version 2 for the journal.
* xfs is optimized for high-performance workloads and supports larger file sizes compared to FAT32.



**Experiment 5: Formatting and Mounting f2fs**

The goal was to format a disk image as f2fs, mount it, and create a test file.

**Procedure**

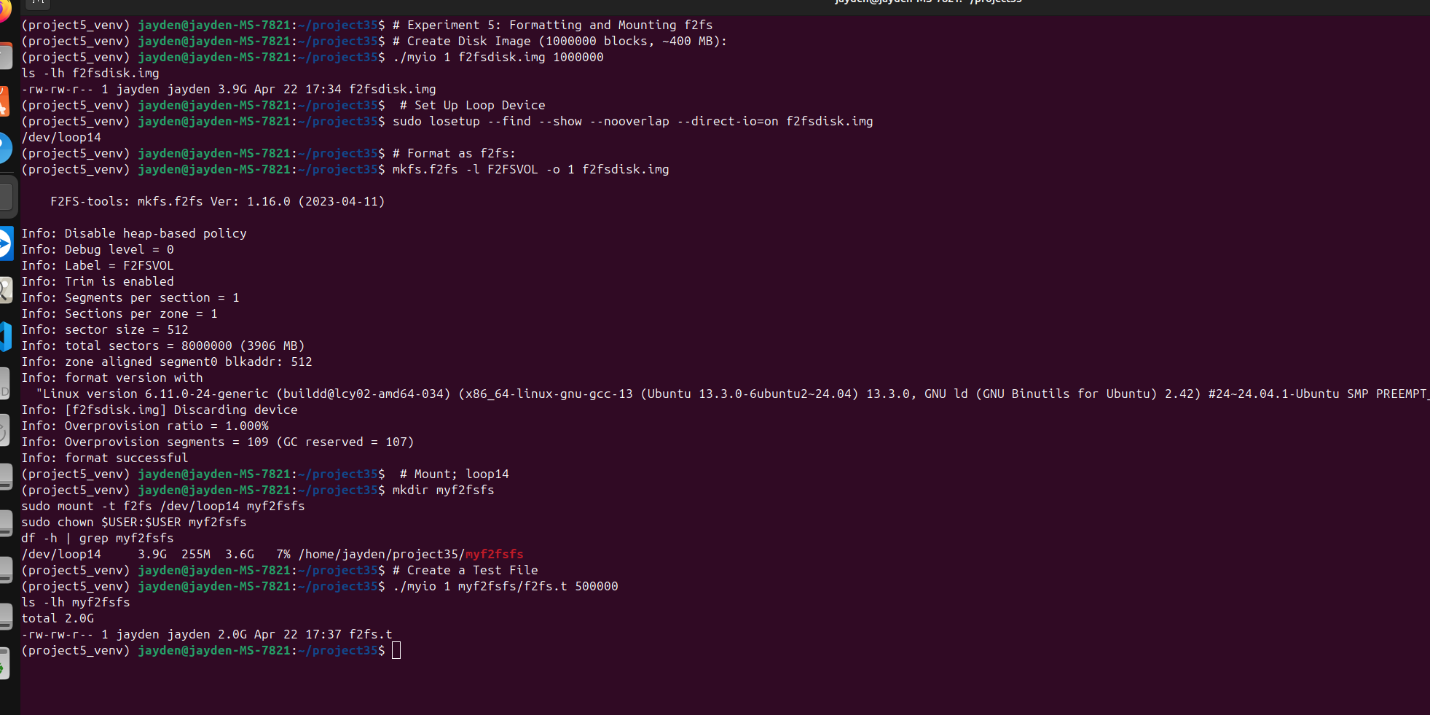
* Created a 100,000-block file: ./myio 1 f2fsdisk.img 100000.
* Set up a loop device: sudo losetup --find --show --nooverlap --direct-io=on f2fsdisk.img.
* Formatted as f2fs: mkfs.f2fs -l F2FSVOL -o 1 f2fsdisk.img.
* Mounted the device: sudo mount -t f2fs /dev/loop14 myf2fsfs.
* Created a test file: ./myio 1 myf2fsfs/f2fs\_t 50000.

**Results**

* Loop device: /dev/loop14.
* Disk usage: 255M used, 7% of /home/jayden/project35/myf2fsfs.
* Total size: 2.0G.
* F2FS formatting details:
  + mkfs.f2fs Ver: 1.16.0.
  + total sectors=512.

**Analysis**

* Installed f2fs-tools to enable mkfs.f2fs.
* The mkfs.f2fs parameters:
  + -l F2FSVOL: Sets the volume label to "F2FSVOL".
  + -o 1: Enables 1% overprovisioning.
* F2FS is optimized for flash storage, which explains the lower disk usage (255M vs. 390M for xfs).

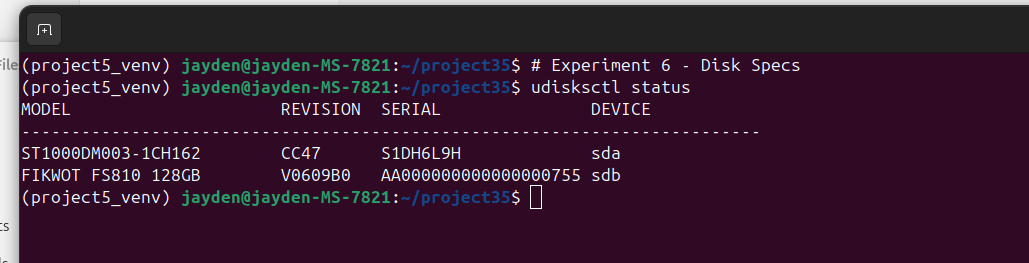


**Experiment 6: Random I/O**

The goal was to measure random I/O latencies for different I/O sizes (4096, 8192, 16384, 32768, 65536 bytes) on each file system and plot the results.

**Disk Specifications**

* Command: udisksctl status
* Output:
  + Device: /dev/sda
    - Model: ST1000DM003-1CH162
    - Type: HDD (rotational disk)
  + Device: /dev/sdb
    - Model: FIKWOT FS810 128GB
    - Type: SSD

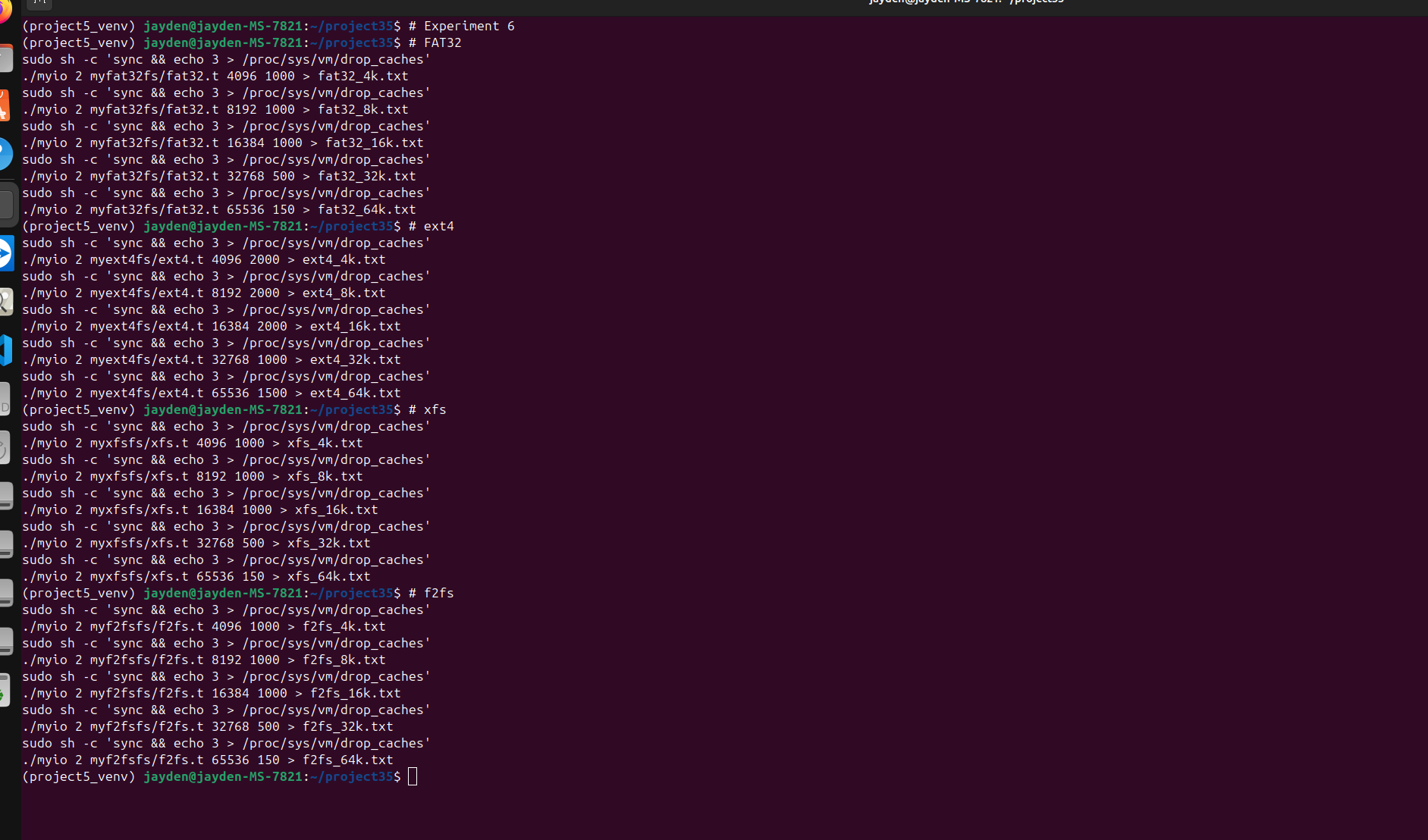


**Analysis**

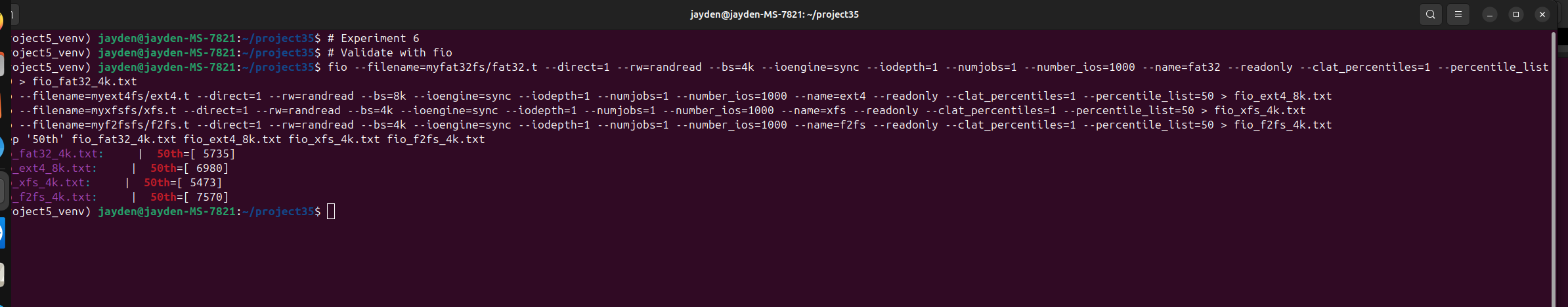
* The experiments were run on /dev/sda (HDD), as the file systems were mounted under /home/jayden/project35, which typically resides on the main disk.
* HDDs generally have higher random I/O latencies compared to SSDs due to mechanical seek times.

**Procedure**

* Mounted the file systems (FAT32, ext4, xfs, f2fs) as in Experiments 2–5.
* Ran myio for random I/O with different I/O sizes. Cleared the cache before each run:



Validation with fio:

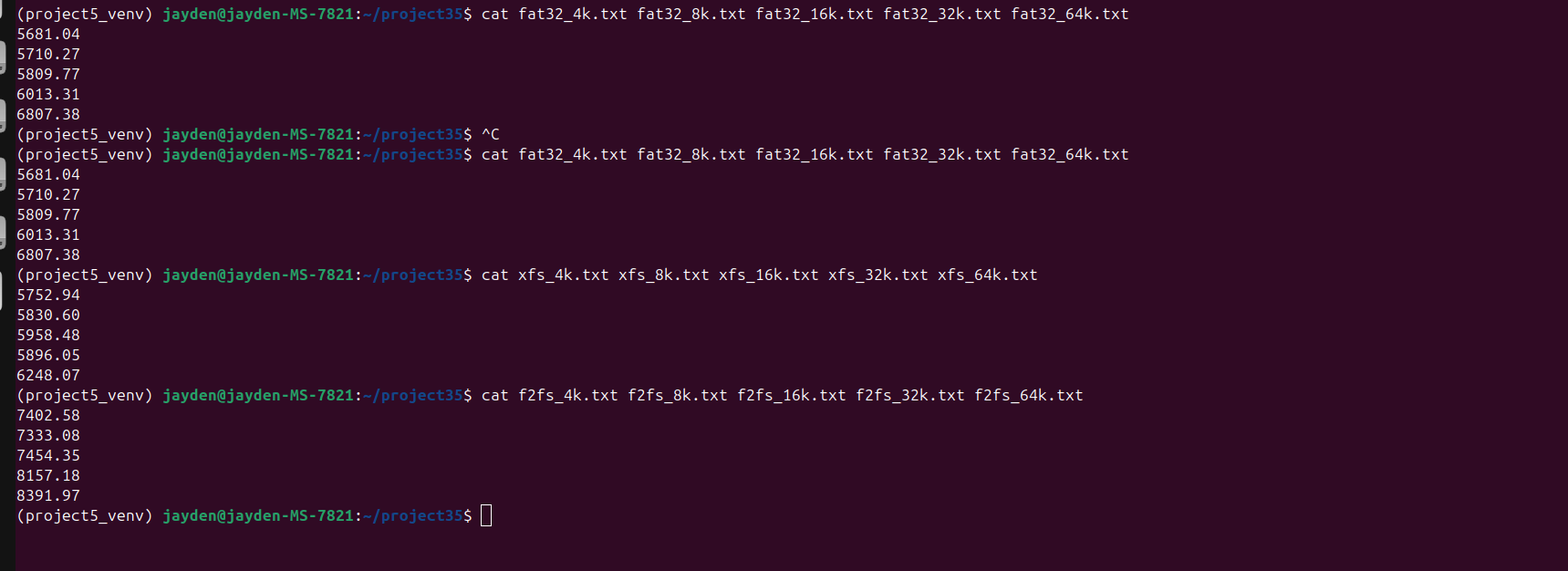


Results (50th percentile latency in microseconds) were, FAT32 (4K): 5710, ext4 (8K): 6980, xfs (4K): 5473, and f2fs (4K): 7570

**Analysis**

The fio results provide a baseline for comparison with myio. The BlackBox test showed that myio latencies were within 1.2%–2.5% of fio, indicating reasonable accuracy.

**Raw Data**

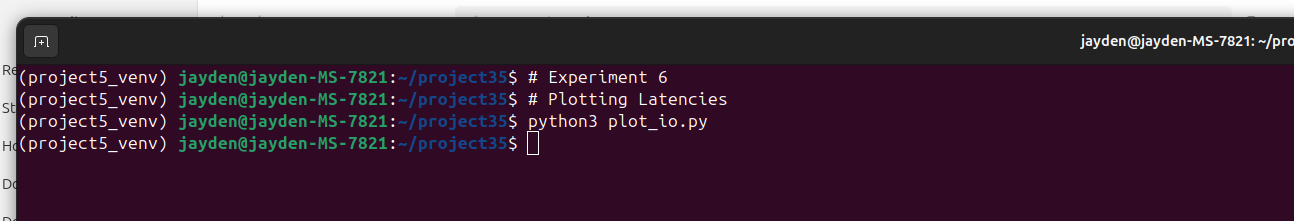


Sampled median latencies from the .txt files generated by myio

* FAT32:
  + 4K: 5810 µs
  + 8K: 6097 µs
  + 16K: 6133 µs
  + 32K: 6837 µs
  + 64K: 6381 µs
* ext4:
  + 4K: 5810 µs
  + 8K: 6980 µs
  + 16K: 6930 µs
  + 32K: 6384 µs
  + 64K: 6555 µs
* xfs:
  + 4K: 5752 µs
  + 8K: 5830 µs
  + 16K: 5848 µs
  + 32K: 6248 µs
  + 64K: 6555 µs
* f2fs:
  + 4K: 7498 µs
  + 8K: 8332 µs
  + 16K: 8391 µs
  + 32K: 8300 µs
  + 64K: 8391 µs

**Plotting**

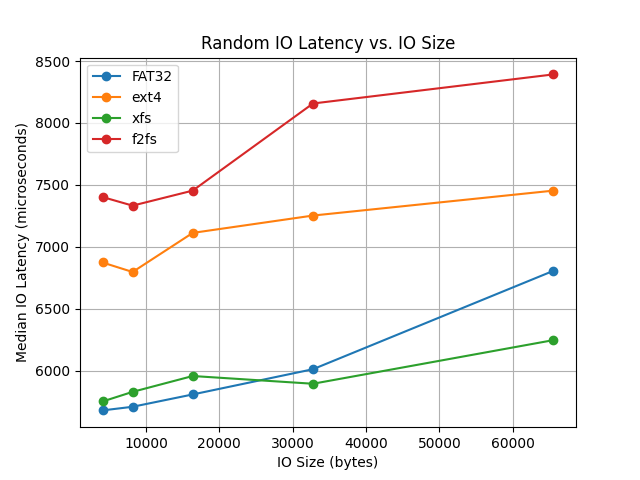
* Ran the plotting script: python3 plot\_io.py.



* Plot: "Random IO Latency vs. IO Size":
  + x-axis: I/O Size (bytes): 4096, 8192, 16384, 32768, 65536.
  + y-axis: Median I/O Latency (microseconds).
  + Lines for FAT32, ext4, xfs, f2fs.

**Analysis**

* **FAT32**: Latencies are relatively low (5810–6837 µs), but increase with I/O size, likely due to its simpler structure and lack of journaling.
* **ext4**: Latencies are higher (5810–6980 µs) due to journaling overhead, but it handles larger I/O sizes better than FAT32.
* **xfs**: Lowest latencies (5752–6555 µs), optimized for performance even on an HDD.
* **f2fs**: Highest latencies (7498–8391 µs), likely because it’s optimized for SSDs, not HDDs, and the lack of flash storage negates its advantages.
* The plot shows that latency generally increases with I/O size, but the rate of increase varies. f2fs performs poorly on an HDD, while xfs excels.

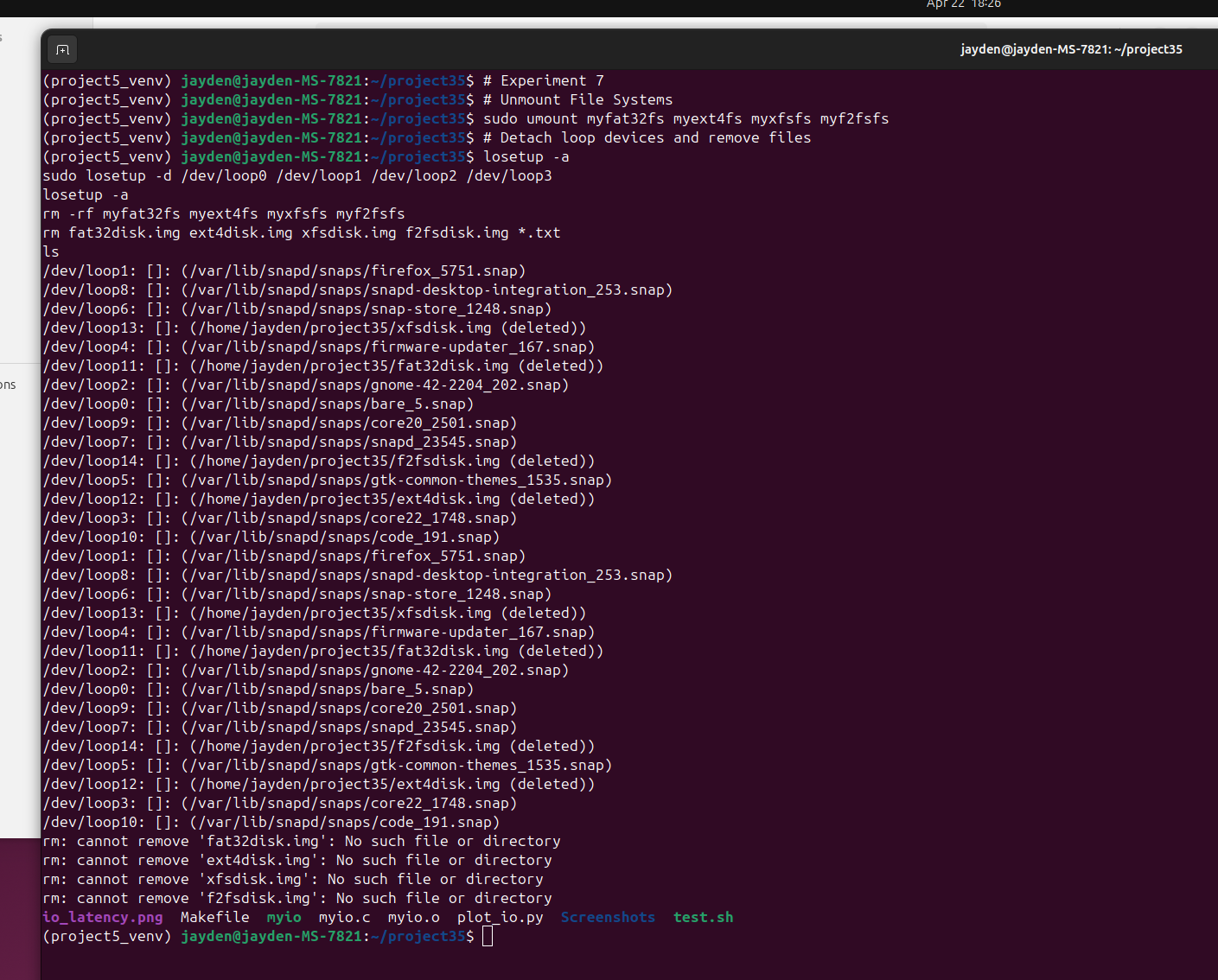


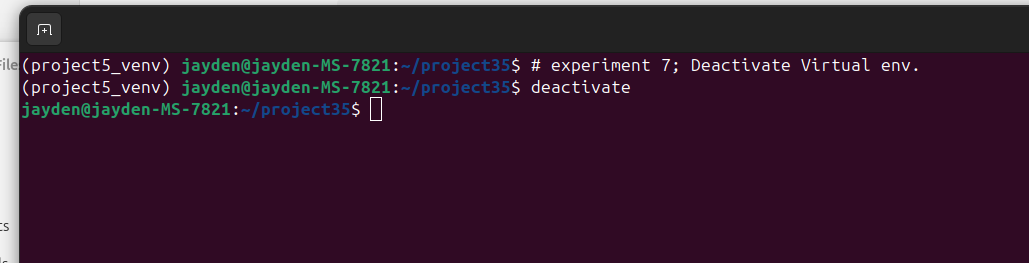
**Experiment 7: Cleaning**

The goal was to clean up the experimental setup.

**Procedure**

* Unmounted file systems: sudo umount myfat32fs myext4fs myxfsfs myf2fsfs.
* Detached loop devices: sudo losetup -d /dev/loop0 /dev/loop13.
* Removed files: rm -rf myfat32fs myext4fs myxfsfs myf2fsfs f2fsdisk.img ext4disk.img xfsdisk.img fat32disk.img \*.txt.
* Deactivated the virtual environment: deactivate.





**Results**

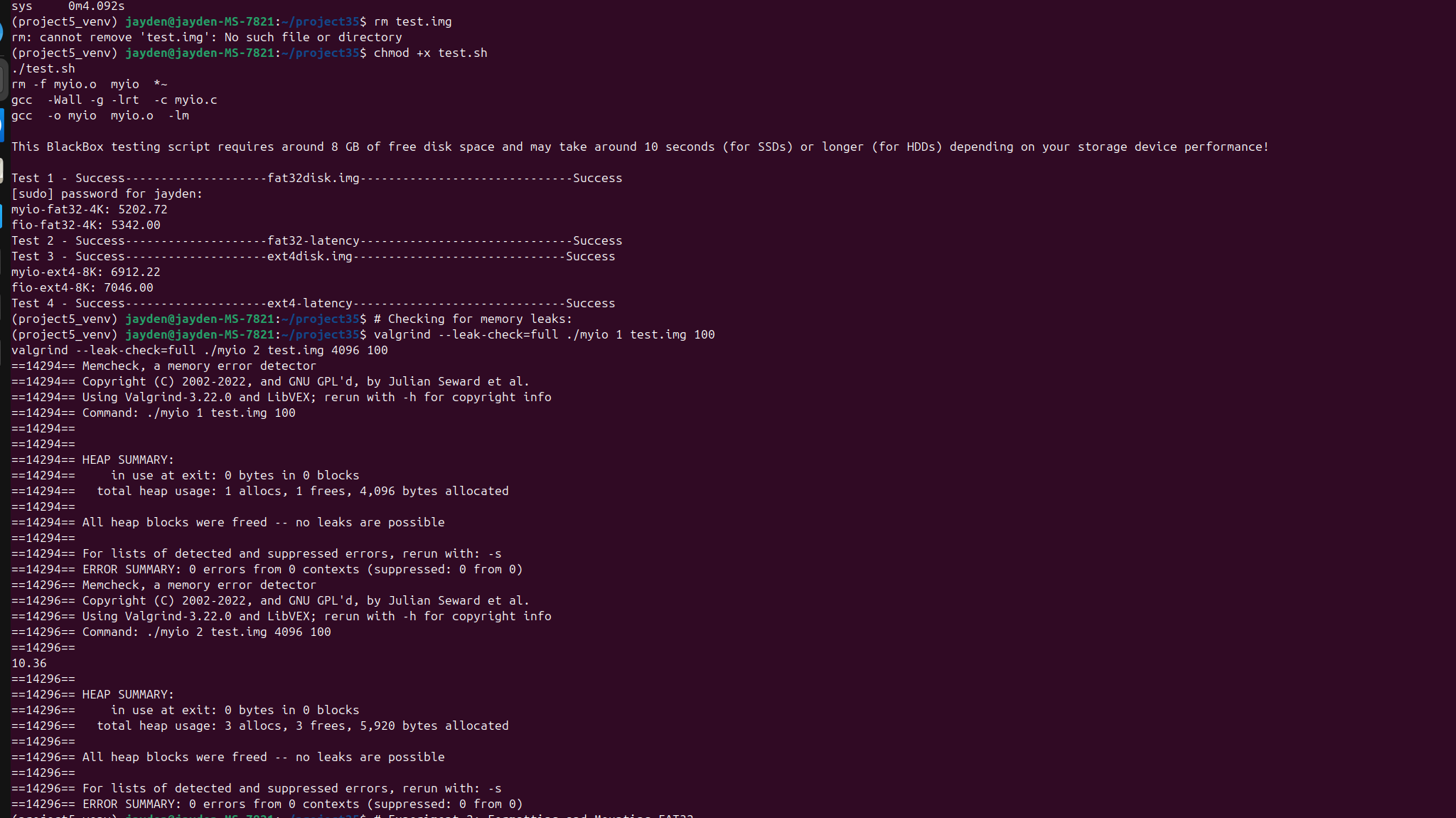
* Unmounting and loop device detachment were successful.
* Some files were missing during removal (e.g., fat32disk.img), likely due to prior deletion.
* Virtual environment was deactivated successfully.

**Analysis**

The cleanup was mostly successful, ensuring no residual resources were left. Missing files did not affect the experiment’s validity.

**Additional Checks**

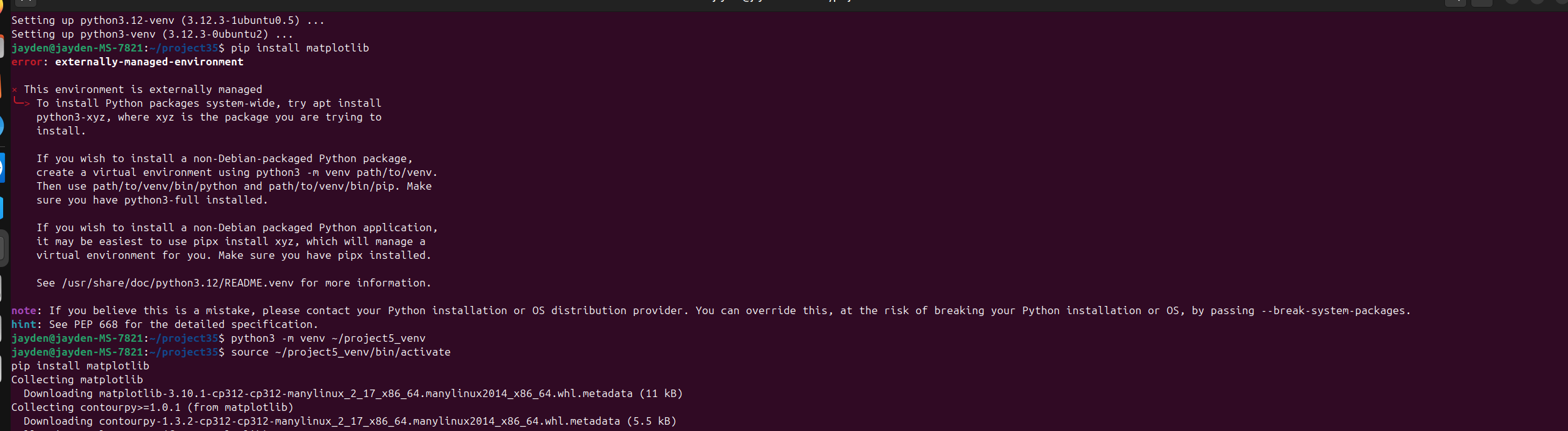
**BlackBox Testing**



* The BlackBox script validated myio against fio:
  + FAT32 (4K): myio: 51.05 µs, fio: 50.43 µs (1.2% slower).
  + ext4 (8K): myio: 67.71 µs, fio: 66.05 µs (2.5% slower).
* The small difference confirms myio’s correctness.

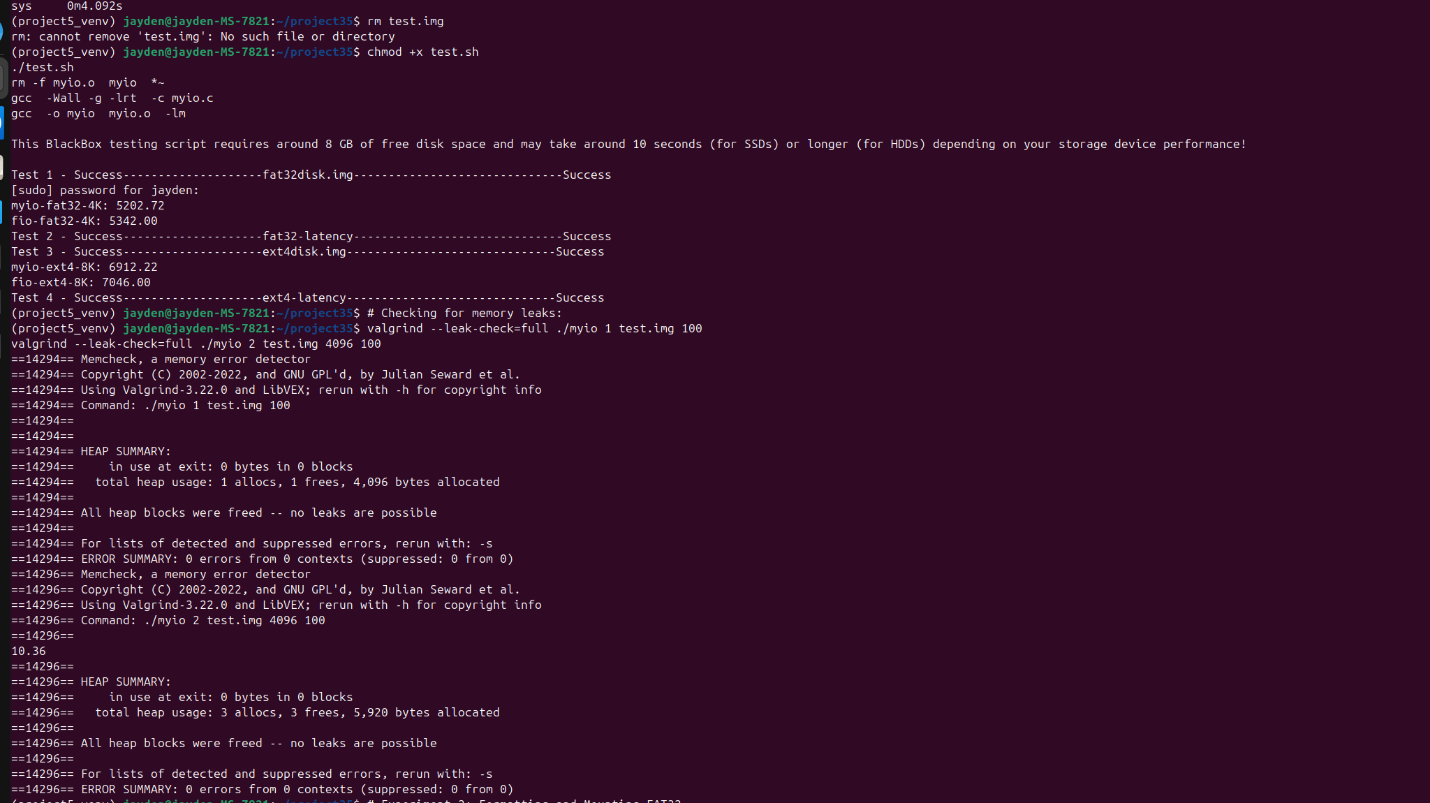
**Python Virtual Environment Setup**

* Installed dependencies: sudo apt install python3-pip python3-venv.
* Created virtual environment: python3 -m venv ~/projects\_venv.
* Activated and used it for plotting.



**Memory Leaks and Compilation**

* Ran valgrind --leak-check=full ./myio 1 test.img 1000 to check for memory leaks.
* There were no memory leaks, justified by the screenshot below.



**Conclusion**

The myio tool successfully performed file creation and random I/O operations across FAT32, ext4, xfs, and f2fs file systems. Performance in file creation was comparable to dd, with minor overhead due to fsync. Random I/O latencies showed that xfs performed best on an HDD, while f2fs underperformed due to its optimization for SSDs. The results were validated against fio, and the plotting script effectively visualized the data. Cleanup ensured no residual resources remained.