Submitted by Mridul Mittal
N13658316
mm13171
mridul.mittal@nyu.edu

Assignment 2 Part 3 Writeup

I realized there were several shortcomings in my original solution:

- Exception Handling: I was catching and suppressing all exceptions within the readat, writeat, and close methods. This is an error as this would suppress any genuine exception.
- Buffering Strategy: My buffering strategy, which stored data and offsets in pending_data and pending_offset, was simplistic. I didn't account for the file's state after multiple write operations or the size of the file, which could lead to incorrect behavior if an undo was called.

In my revised solution, I recognized these errors and made several improvements:

- Error Reporting: Instead of suppressing exceptions, I allowed them to propagate up to the caller. This way, users of the system are informed about any issues and can handle them appropriately. This change also meant removing the try-except blocks, which were silently catching all exceptions.
- State Management: I added attributes to track the state of the file more effectively. Specifically, I introduced filesize to keep track of the current size of the file. This was necessary because I needed to know whether a write operation would extend past the end of the file, which would be an error if the buffer hadn't been flushed yet.
- Locking: I incorporated a lock using createlock(), and used it to protect critical sections of the code. This was a significant enhancement, ensuring that my LPFile methods could be used safely in a concurrent environment, preventing race conditions and ensuring consistency.
- Flush Buffer Logic: I added a boolean flushBuffer to indicate whether the buffer contains data that needs to be written to the file. This provided a clear mechanism to check whether there is buffered data that either needs to be written (upon a new writeat call or close) or discarded (upon undo).
- Deferred Writing: In the writeat method, I deferred writing new data to the file if there was already data in the buffer. If flushBuffer was True, indicating that the buffer was empty, I could safely write the data to the file. Otherwise, I wrote the data that was in the buffer first, before buffering the new data. This logic ensured that undo would only affect the most recent write operation.

Through these iterations, I addressed the critical flaws in my original implementation. I enhanced the system's reliability and security, aligning it with the assignment's specifications, and I learned valuable lessons about system design and error handling in a security context.