Problem: Should have thrown a ….. But RepyArgumentError was thrown. So in my Try Block I was trying to catch exceptions in my code and handle them by raising a RepyArgumentError. However, the problem was that I was consistently raising the RepyArgumentError in my except block, which meant that it would be raised regardless of the specific exception that occurred.
How I fixed it: In order to resolve this error, I needed to implement exception handling that is tailored to the specific error that is encountered. The RepyArgumentError should only be raised if and when that particular error occurs. I have incorporated try-except-finally blocks in my code. This method is designed to catch exceptions, re-raise them without any specific handling, and guarantee the execution of code within the finally block, regardless of whether an exception occurred.
Problem: It the write() Function: it permits writing to a negative offset, which can lead to a vulnerability if an attacker calls close() and attempts to write beyond the end of the file.
How I fixed it: I used conditional statements to check if an offset is negative, then raise "RepyArgumentError". I added a global variable to keep in track if the file is closed and initialized it to false. Then, I implemented a check to see if the value of "self.file_closed" is true, which is set when the close() function is called. If it is true, I raise a "FileClosedError" to indicate an attempt to write to a file that has already been closed. Last, I initialize a global variable that calculates the size of the file, initializing a waiting file size, and then checking if the provided offset exceeds the waiting/pending file size. If it does, it raises a custom exception, SeekPastEndOfFileError, to handle situations where attempting to seek or retrieve data beyond the end of a file is detected.
Problem: Ensuring the file size is updated correctly to maintain an accurate record of the file's size, particularly when invoking the writeat() or undo() functions.
How I fixed it: Now since I keep track of the file size, I have to implement updates to the self.pending_file_size to match the expected size of the file after appending data. This will ensure that it reflects the expected size of the file after new data is appended, particularly when writing data beyond the current known file size. Also, it is necessary to modify the pending/waiting file size following the execution of the undo() and close() functions, in order to represent the size of the file once the pending data has been written.
Problem:The problem at hand involves the use of threading, where multiple threads attempt to at the same time access and modify the same data. Also, race conditions are situations that arise when the result of a program is influenced by the order and timing of events that occur between different threads
How I fixed it: Implementing locks so only one thread can access it at a time and preventing race conditions. I have already incorporated locks into my initial reference monitor. However, I have made some modifications to ensure their appropriate usage in my code. I call `self.locked.acquire(True)` to request exclusive access to a critical section of code. If the lock is currently accessible, the thread will obtain it, enabling it to continue its execution. Then using the self.locked.release() to release control of the lock, allowing other threads that are waiting to access it.