Due to a misunderstanding, I originally assumed all errors should only be thrown on write commits, thus giving me a fair amount of work fixing up my reference monitor. The first, and easiest, fix, was to throw a RepyArgumentError whenever writeat was called with a negative or non-integer offset, or with non-string data.

I added an is_open boolean, initialized as True under the LPFile constructor, to keep track of file state. Writeat and close check this boolean on every operation and throw a FileClosedError if false. Close also writes any valid pending data to the file before setting the is_open boolean to false and closing the file. Undo will not execute if the file is already closed. This avoids writing to or closing already closed files.

Keeping track of the file length was a bit trickier. I included a file_length attribute, initialized to the length of the opened file in the constructor. I also save a prev_length attribute, initialized to the same length in the constructor. If the sum of the offset and the length of the data is greater than or equal to the current file_length, prev_length is set to the current file_length and file_length is updated to offset + len(data) in writeat. Updating file_length in this fashion avoids having to call readat, which would slow down the reference monitor's performance. The equality condition catches a corner case where, e.g., two writes of equal length are called after opening an empty file, then undo is called. If the file length only updated when the offset + the length of the pending data were strictly greater, undo would erroneously update file_length back to 0. Keeping track of prev_length allows undo to function correctly when 2 writes extend the length of the file, but the second write overwrites some part of the preceding write which extended beyond the initial length. RepyArgumentError, FileClosedError, SeekPastEndOfFileError are all thrown in order of priority as spelled out in the repy docs.

I also included a write lock for safe, concurrent functionality. The write lock is used on writeat, undo, and close, as none of these functions should be able to run in parallel, but this allows readat to be called in parallel and increase the reference monitor's performance. The locks are acquired, and all following functionality is encapsulated in a try block with lock release guaranteed in a finally block. Note that the write call in close is a direct API call to avoid deadlock.

As a sidenote, I offer the suggestion that the next iteration of this assignment include mandatory, succinct commenting, and the possibility of throwing out testcases with unclear comments or an unnecessary level of test complexity (e.g. a dozen successive writes when 2 or 3 would catch the relevant corner case). Even as a toothless threat, this may encourage best practices. I believe this would help both the students and the grading team save a fair amount of time interpreting some of the worst code etiquette offenders.