Assignment 2 Part 3

The initial reference monitor I designed was vulnerable to several issues, particularly related to concurrency and the management of uncommitted writes. These vulnerabilities were addressed and fixed using semaphores and end-of-file (EoF) tracking mechanisms, leading to a more robust final result.

Vulnerabilities in the Initial Reference Monitor:

- Concurrency Issues:
  The initial reference monitor did not provide synchronization mechanisms to control concurrent access by multiple threads to the internal state of the LPFile class. This made the monitor susceptible to race conditions, where multiple threads could interfere with each other's operations, leading to unpredictable results.

- Uncommitted Writes:
  The initial reference monitor did not account for the management of uncommitted writes. A write operation didn't immediately write its changes to the file, and there was no clear mechanism to handle uncommitted writes. This could result in data inconsistencies and confusion regarding which changes were permanent and which were not.

Fixes Using Semaphores and EoF Operations:

- Concurrency Control with Semaphores:
  Semaphores were introduced to control concurrent access to critical sections of the LPFile class, such as read and write operations. These semaphores ensured that only one thread could access the file's internal state at a time.

  The readat and writeat methods were enclosed in semaphore-protected regions. This prevented race conditions and ensured that operations were executed sequentially.

- EoF Tracking Mechanism:
  To handle uncommitted writes, the reference monitor was given an EoF tracking mechanism, keeping track of the old EoF and pending EoF positions.

  When a write operation was executed, it was first stored in the self.pending_data attribute, and the corresponding EoF updates were synchronized within a critical section protected by semaphores.