Report: Comparative Analysis of Old and New Reference Monitors

In this report, we delve into a comparative analysis between the previous version of the reference monitor and its upgraded counterpart, focusing on six essential modules: LPopenfile, init, readat, writeat, undo, and close.

We begin with the LPopenfile module. Previously, the method employed was to directly use the split function to dissect the string and ascertain the legality of the file name. However, this posed a challenge; if the string failed to adhere to the specified splitting rules, it would trigger an array out-of-bounds error. Additionally, when the create parameter was set to False, it was essential to verify the existence of the file rather than initiating a new file creation. The revised reference monitor addresses these issues adeptly. Initially, it checks the successful splitting of the string, followed by an assessment of the file naming conventions, and finally, it examines the existence of the file, and based on the create parameter, decides whether a new file needs to be created. Should any check fail, it throws an error, intercepting any illicit operations.

Moving on to the LPfile.init module, the prior reference monitor overlooked potential errors that might occur during the file opening process. In contrast, the new version acknowledges the possibility of file opening failures, and has been enhanced with internal variables for caching and locking. Here, the lock is employed for write-locking, while buffer and length are utilized for caching the file content and its length respectively.

In the upgraded LPfile.readat module, a mechanism to validate the legitimacy of parameters has been introduced. For the offset parameter, it's imperative to check for any out-of-bounds conditions, reporting an error if encountered. Similarly, for the bytes parameter, an out-of-bounds check is conducted, along with a check for a None value. If a None value is found, all subsequent data from the offset is returned.

The new LPfile.writeat module now incorporates a locking operation at the outset to mitigate thread contention during simultaneous writes by multiple threads. Following this, a check is conducted to ascertain the legality of the offset, bifurcating scenarios based on the presence or absence of pending_data. If pending_data is present, it is first written into the file, with the data being stored in pending_data thereafter.

Enhancements in the LPfile.undo module include the integration of a thread lock to avert thread contention, a feature mirrored in the LPfile.close module to prevent premature file closure during ongoing write operations.

With these aforementioned enhancements, our reference monitor has successfully cleared all test scenarios, barring those under review or deemed invalid.