Christopher Torres
CS-UY 3923
Justin Cappos
Security Layer Enhancement Report - 11/02/2023

The initial implementation of the security layer allowed for file operations with a limited capability to undo write actions. While it provided a foundational structure, there were vulnerabilities and inefficiencies that led to potential data corruption, race conditions, and memory consumption issues. This report dives into the vulnerabilities identified from my classmate's attacks, and how I implemented them.

Vulnerabilities Identified:

- ➔ Single-level Undo Limitation: The previous version only supported undoing the last write operation, which could lead to data loss if multiple changes needed to be reverted.
- ➔ Concurrent Access: Without locking mechanisms, concurrent access to file operations could lead to race conditions, potentially corrupting data.
- ➔ Lack of Offset-Specific Undo: An undo operation reverted the entire file to its previous state, making it inefficient and less flexible.
- ➔ Unbounded Undo History: Without a limit on undo history, the system could consume excessive memory when dealing with large files or frequent write operations.
- ➔ Inadequate Error Checks: There were insufficient boundary and state checks, potentially leading to erroneous operations like negative offset writes.

Enhancements and Their Rational:

- ➔ Multi-level Undo: By transitioning from a single shadow copy to an `undo_dict` structure, the system can now keep track of multiple write operations for each offset. This gives the user greater flexibility to revert specific changes without affecting other parts of the file.
- ➔ Locking Mechanism: Introducing the `createlock()` function ensures that file operations are thread-safe. The `acquire()` method with a timeout prevents potential deadlocks, ensuring that the system remains responsive. This addresses potential concurrency issues and guarantees data integrity.
- ➔ Improved Undo Stack: The `undo_dict` provides a more granular approach to track changes. Each offset has its list of data changes, ensuring efficient and specific undos.
- ➔ Boundary Checks: By checking offsets for validity, such as ensuring non-negative values, the system prevents logically incorrect operations, enhancing the robustness of the layer.
- ➔ Limit on Undo Stack Size: The introduction of `MAX_UNDO_SIZE` sets a cap on the number of undos stored for each offset. This strikes a balance between offering the undo functionality and managing memory usage effectively.
- ➔ Clearing Undo History on Close: By clearing the undo history when a file is closed, the system ensures no carry-over of undo operations between different file sessions, preventing potential data inconsistencies.

The enhanced security layer provides a robust framework for file operations, balancing functionality with security and efficiency. By addressing vulnerabilities related to concurrency, data integrity, and memory management, the system allows it to be secure against the majority of test cases provided by my peers.