Assignment 2.3

Vulnerability:

The reference monitor did not use locks. This made it vulnerable to race conditions while performing multithreaded operations.

Solution:

Python threading locks were used to control access to the file while read, write, undo, and close operations. Special care was taken to release the lock in case an error was raised while performing these operations.

Example of updated code:

Previous version:

```
def writeat(self, data, offset):
        if self.file_closed or offset < 0 or offset > self.pending_file_size:
            self.LPfile.writeat(data,offset)

        if self.pending_data is not None and self.pending_offset is not None:
            self.LPfile.writeat(self.pending_data, self.pending_offset)
            self.filesize = self.pending_file_size
        self.pending_data = data
        self.pending_offset = offset
        self.pending_file_size = max(self.pending_file_size,self.pending_offset + len(self.pending_data))
```

Updated version:

```
def writeat(self, data, offset):
        self.lock.acquire(True)
        try:
            if self.file_closed or offset < 0 or offset > self.pending_file_size:
                self.LPfile.writeat(data,offset)

            if self.pending_data is not None and self.pending_offset is not None:
                self.LPfile.writeat(self.pending_data, self.pending_offset)
                self.filesize = self.pending_file_size
            self.pending_data = data
            self.pending_offset = offset
            self.pending_file_size = max(self.pending_file_size,self.pending_offset + len(self.pending_data))
        finally:
            self.lock.release()
```