Notexio Project Report

Course: CSE323 (Operating Systems Design)
Date: December 20, 2025
Project: Notexio Text Editor

1. Introduction

Notexio is a comprehensive text editor developed as a final project for CSE323. The goal was to build a functional, user-friendly application that demonstrates core Operating Systems design concepts such as File I/O handling, process management, memory buffering, and event-driven concurrency. Built with Python and Tkinter, Notexio replicates the core functionality of standard system editors like Notepad while adding advanced features like auto-save recovery, theming, and real-time statistics.

2. Challenges and Solutions (STAR Format)

Challenge 1: Data Persistence and Crash Recovery

Situation: A critical requirement for any text editor is data safety. During testing, it became apparent that system crashes or accidental closures would result in significant data loss for the user.

Task: The objective was to implement an automatic 'background' saving mechanism that persists the user's work without freezing the user interface or requiring manual intervention.

Action: I implemented a threaded recovery system (SafetyFeatures class).
- I used Python's threading module to create a daemon thread that runs parallel to the main application loop.
- This thread monitors the is_modified flag and saves a snapshot of the current text to a hidden recovery/ directory every 5 minutes.
- I had to ensure that file write operations (I/O) did not block the main UI thread, effectively separating the 'worker' process from the 'interaction' process.

Result: The application now features a robust recovery system. Upon startup, it scans the recovery directory and prompts the user to restore any unsaved work found, effectively mitigating data loss risks.

Challenge 2: Synchronized Line Numbering

Situation: Users required line numbers for code editing. However, Tkinter's default Text widget does not support built-in line numbers, and simply drawing them on a canvas often leads to misalignment during scrolling.

Task: I needed to create a sidebar that displays line numbers which scroll in perfect synchronization with the main text content.

Action: I implemented a 'dual-widget' architecture in UIComponents.
- I placed a secondary, read-only Text widget to the left of the main editor.
- The technical challenge was capturing the scroll event correctly. I bound the yview command of the line-number widget to the scrollbar of the main widget.
- I also listened for the <> event to recalculate line numbers dynamically whenever the user adds or removes lines.

Result: The result is a seamless, professional-looking editor where line numbers track perfectly with the content, regardless of scroll speed or font size changes.

Challenge 3: Efficient Real-time Statistics (Event Handling)

Situation: The requirement was to show real-time word, character, and line counts in the status bar. Initially, recalculating these statistics on every keystroke caused noticeable input lag, especially in large documents.

Task: The task was to optimize the event handling to update the UI without degrading performance.

Action: I optimized the event loop interaction.

- Instead of triggering a full document scan on every Key event, I utilized KeyRelease and debouncing techniques.
- I also leveraged the efficient internal index system of the text widget to retrieve counts rather than iterating through the string manually.

Result: The status bar now updates strictly when necessary, maintaining a smooth 60fps typing experience even with documents containing thousands of words.

3. Technical & Theoretical Analysis

File I/O and Buffering

The project relies heavily on the OS's file system interface. When the FileManager opens a file, it requests a file descriptor from the OS. A key theoretical concept applied here is **buffering**. When reading files, we use Python's built-in buffered I/O (via open()) to minimize the number of expensive system calls. This ensures that even if the disk is slow, the application reads data in efficient chunks (pages) rather than byte-by-byte.

Concurrency and Threading

The auto-save feature demonstrates the concept of **Concurrency**. The main application runs on the 'Main Thread' which handles the GUI event loop. If we were to perform file writing on this thread, the OS would pause the application window until the disk write is complete (blocking I/O). By spawning a separate daemon thread for the SafetyFeatures module, we allow the OS to schedule these CPU cycles independently, ensuring the UI remains responsive (non-blocking).

4. Conclusion

Developing Notexio was a practical exercise in applying Operating Systems theory to software development. By solving problems related to process concurrency, I/O management, and resource optimization, the project successfully delivered a stable and feature-rich text editor. The final product meets all functional requirements and provides a safe, efficient environment for text editing.