Notexio Project Report

CSE323 - Operating Systems Design

1. Introduction

Notexio is a comprehensive, lightweight text editor built using Python and Tkinter. Designed as a final project for the Operating Systems Design course, it demonstrates practical applications of core OS concepts including File I/O management, event-driven programming, and background process handling (threading). Notexio provides a user-friendly interface with essential features such as syntax highlighting (via themes), auto-saving, and recovery mechanisms, bridging the gap between theoretical OS principles and real-world software development.

2. Project Highlights

- Robust File System Interaction: Implements safe reading, writing, and modifying of files with proper error handling and permission checks.
- Customizable UI: Features a dynamic theme manager allowing for Light, Dark, and Custom modes.
- Safety First: Includes an automated background recovery system that protects user data against crashes.
- Rich Editing Features: Supports undo/redo stacks, line numbering, find/replace, and text formatting.

3. Challenges and Solutions (STAR Format)

Challenge 1: Synchronized Line Numbers

Situation:

Users require visual cues (line numbers) to reference specific parts of the text. However, the standard Tkinter Text widget does not provide built-in line numbering, and creating a static side panel leads to misalignment when scrolling.

Task:

Implement a dynamic line number bar that stays perfectly synchronized with the main editor's scrolling and content updates.

Action:

I created a separate Text widget solely for displaying line numbers and placed it adjacent to the main editor. I disabled user input on this widget to prevent accidental editing. To ensure synchronization, I bound scroll events to hook into the scrollbar command and mouse wheel events to scroll both widgets simultaneously. I also attached listeners to the <<Modified>> event of the main editor to update the line counts dynamically.

Result:

The editor features a professional, non-intrusive line number sidebar that updates in real-time and scrolls seamlessly with the document, enhancing code readability and navigation.

Challenge 2: Implementing a Dynamic Theme System

Situation:

Hardcoding colors limits usability, especially in low-light environments where 'Dark Mode' is essential. Changing colors at runtime usually requires restarting simple Tkinter apps or complex state management.

Task:

Create a centralized ThemeManager that allows users to switch themes instantly without restarting the application, updating all UI components (menus, status bar, text area) dynamically.

Action:

I designed a ThemeManager class that maintains a dictionary of color palettes. I implemented an apply_theme method that traverses the widget hierarchy, identifying components and applying the corresponding colors. I also integrated this with a SettingsManager to persist the user's preference in a JSON configuration file.

Result:

Notexio supports instant theme switching. Users can toggle between Light and Dark modes effortlessly, improving accessibility and user experience.

Challenge 3: Background Auto-Save and Crash Recovery

Situation:

One of the critical responsibilities of an OS is data integrity. Standard text editors often lose unsaved data if the process terminates unexpectedly.

Task:

Implement a non-blocking auto-save feature that periodically backups work without freezing the user interface.

Action:

I utilized Python's threading module to create a separate thread for the auto-save operation. The SafetyFeatures class spawns a daemon thread that wakes up every 5 minutes, checks the is_modified flag, and writes content to a timestamped file in a dedicated recovery/ directory. On startup, the system scans this directory for recovery files.

Result:

The application mimics OS-level journaling systems, ensuring that user data is safe even during catastrophic failures, while the main UI thread remains responsive.

4. Technical Concepts

Event-Driven Programming

Notexio relies heavily on the event loop provided by the window manager. User actions generate events that are dispatched to specific handler functions, ensuring immediate feedback for interactions.

File I/O and Exception Handling

The FileManager module handles all disk interactions. It uses try-except blocks to gracefully handle errors

such as 'File Not Found' or 'Permission Denied,' ensuring robustness.

Threading

To prevent the 'Application Not Responding' state during auto-saves, I delegated the file writing task to a background thread. This demonstrates the OS concept of concurrency.

5. Conclusion

Notexio successfully meets the requirements of a modern text editor while serving as a practical implementation of Operating Systems Design concepts. Through solving challenges related to process synchronization, resource management, and UI responsiveness, the project delivers a stable and efficient tool for text manipulation.