Notexio - Project Report

CSE323: Operating Systems Design

1. Project Introduction

Notexio is a lightweight, customizable text editor built using Python and Tkinter. It was designed to demonstrate operating system concepts such as file management, process handling (via the GUI event loop), and memory management (buffer handling for text). The application provides essential text editing features including file operations (New, Open, Save), editing tools (Undo/Redo, Find/Replace), and customization options (Themes, Fonts).

2. Challenges Faced (STAR Analysis)

Challenge 1: Synchronized Line Numbers

Situation: One of the key requirements for a code-friendly text editor is line numbering. However, the standard Tkinter Text widget does not provide a built-in line number feature.

Task: The task was to implement a line number sidebar that automatically updates when lines are added or removed and synchronizes its scrolling with the main text area.

Action: I created a separate Text widget to serve as the line number bar. I disabled standard user input on this widget to prevent manual editing. I then bound event listeners to the main text widget's '<<Modified>>' event and scroll commands. A function was written to calculate the number of lines in the main text and update the sidebar accordingly. I also implemented a custom scroll handler to ensure both widgets scroll simultaneously.

Result: The result is a seamless line numbering system that stays in sync with the user's content, enhancing the editing experience for code and structured text.

Challenge 2: Robust Find and Replace System

Situation: Implementing a search and replace feature involves handling various edge cases, such as case sensitivity, search direction (forward/backward), and 'Replace All' functionality without freezing the UI on large files.

Task: I needed to build a dialog-based interface that allows users to find text and optionally replace it, providing feedback when matches are found or not.

Action: I utilized tk.Toplevel to create non-blocking dialog windows. For the search logic, I used the text widget's built-in search capabilities but wrapped them in a custom controller to handle 'Find Next' and 'Find Previous'. For 'Replace All', I optimized the process by retrieving the entire text content, performing a regex-based substitution (handling case sensitivity flags), and then updating the widget in one go, rather than iterating through UI updates which would be slow.

Result: A responsive and feature-rich Find/Replace tool was successfully integrated, allowing for efficient text manipulation.

Challenge 3: File Safety and Recovery

Situation: Text editors must ensure data persistence. A crash or accidental closure could result in significant data loss for the user.

Task: The goal was to implement a safety mechanism that auto-saves progress or creates recovery files to prevent data loss.

Action: I designed a SafetyFeatures module. This module monitors the 'modified' state of the document. I implemented a system that periodically checks the file status and creates recovery files in a dedicated 'recovery/' directory.

Additionally, I added an 'on exit' check that intercepts the window close event to prompt the user to save if changes are detected.

Result: The application now robustly handles unexpected closures and protects user data, significantly improving reliability.

3. Conclusion

The development of Notexio provided valuable hands-on experience with GUI programming and OS-level file interactions. Overcoming challenges related to widget synchronization and state management reinforced the importance of event-driven architecture. The final product is a functional, user-friendly text editor that meets the course requirements.