Notexio Project Report

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

1. Project Overview

Notexio is a lightweight, feature-rich text editor designed to demonstrate core operating system concepts such as file I/O operations, process management, and thread-based background tasks. Built with Python and Tkinter, it provides a modern, user-friendly interface for text manipulation while implementing robust safety features like auto-recovery.

The project mimics the functionality of standard text editors (like Notepad or TextEdit) but enhances it with developer-focused features such as line numbers, real-time statistics, and theme customization.

2. Challenges & Solutions (STAR Analysis)

Throughout the development of Notexio, several technical challenges were encountered. Below is a detailed analysis of these problems and their solutions using the STAR (Situation, Task, Action, Result) format.

Challenge 1: Application Architecture & Circular Dependencies

Situation: As the codebase expanded from a simple script into a full-fledged application, I split the code into multiple modules (editor.py, file_manager.py, ui_components.py, etc.) to maintain readability. However, this led to complex circular dependency issues. For example, the FileManager needed access to the Editor to read text for saving, while the Editor needed the FileManager to handle 'Open' commands.

Task: The goal was to establish a modular architecture that allowed for clear separation of concerns without tightly coupled, circular references that cause ImportError or recursion limits.

Action: I implemented a central dependency injection pattern within the main NotexioApp class. This class acts as the 'orchestrator.' It initializes all manager classes first (Editor, FileManager, etc.) and then explicitly injects the necessary references. I utilized a self.app reference passed to components, allowing them to access sibling modules via the parent (e.g., self.app.editor.text_widget).

Result: This resulted in a clean, maintainable codebase where modules are independent but can communicate effectively. It made adding new features (like the Theme Manager) straightforward, as they could simply plug into the existing structure.

Challenge 2: Non-Intrusive Auto-Recovery System

Situation: A critical requirement for any text editor is data safety. I needed to ensure that if the application crashed or the computer lost power, the user's unsaved work would not be lost.

Task: Implement an auto-save mechanism that runs periodically without 'freezing' the user interface or interrupting the typing flow.

Action: I leveraged Python's threading module to create a background daemon thread for the auto-save process. Multithreading allows the operating system to schedule the auto-save task independently of the main

UI thread. The SafetyFeatures class spawns a thread that sleeps for a set interval (300 seconds). When it wakes, it checks the is_modified flag. If changes exist, it writes the content to a timestamped file in a recovery/ directory. I also implemented a startup check to scan this directory and prompt the user for restoration.

Result: The application now provides a robust safety net. Users can recover their work after an unexpected closure, and because the save operation happens in a separate thread, the typing experience remains fluid and lag-free.

Challenge 3: Synchronized Line Numbers

Situation: Implementing a line number sidebar is deceptively difficult. A naive approach of redrawing numbers on every keypress causes flickering and severe performance degradation, especially with large files.

Task: Create a line number display that scrolls perfectly in sync with the text and updates efficiently only when necessary.

Action: I created a separate Text widget for the line numbers and disabled it for user input. To handle synchronization, I bound the scroll events of the main text widget to a handler that programmatically scrolls the line number widget (yview_moveto). I optimized the update logic to recalculate line numbers only on specific events (like <<Modified>> or newline insertion) rather than every keystroke.

Result: The line numbers provide a professional polish to the editor. They scroll smoothly alongside the code, improving readability and navigation without impacting application performance.

3. GitHub Submission Details

Repository Structure

The project is organized as follows:
- src/: Contains all source code modules.
- config/: Stores user preferences and settings.
- recovery/: Directory for auto-saved recovery files.
- main.py: The entry point for the application.

Video Demonstration

[Link to Video Demo Placeholder] - Please insert your video link here

Introduction

Notexio is a Python-based text editor built from scratch to explore OS design principles. It features a modular architecture, thread-safe file operations, and a custom UI framework using Tkinter. This project demonstrates practical application of file management, process handling, and event-driven programming.