



*OPERATING SYSTEMS PROJECT*

---

# **FILE MANAGEMENT SYSTEM**

---

BTECH. CSE + CSE-AI  
2022-2026

***Authors:***

Aditya Pillai

Avinaash A

Skanda S Bhatt

Saatvik Prabhu

***Id:***

CS22B1063

CS22B1064

CS22B1067

CS22B2031

*Faculty: Dr. Jaishree Mayank*

# 1 Introduction

The **Multithreaded File Management System** is a client-server-based application designed to facilitate efficient and secure file operations in a concurrent environment. The backend of the system is implemented in C, functioning as the server, which handles the core logic for file operations, including locking and unlocking mechanisms to ensure data consistency and security. The client-side interface, developed in Python, provides a user-friendly GUI to interact with the server. Communication between the client and server is achieved through sockets, enabling seamless requests for file operations such as reading, writing, renaming, and deletion. This system is designed for environments that require high concurrency and robust file management.

## 2 Purpose

The Primary Purpose is to create a clone of the file explorer found in every operating system. To imitate the way it works so as to understand how the reader writer problem is extended and tackled by the modern operating systems.

## 3 Intended Audience and Reading Suggestions

This document is intended for requirements engineers, students and professors of an Institute, technical engineers. Reading Suggestions

## 4 Document Conventions

The document focuses on the high priority requirements which will be implemented for the final deliverable.

## 5 Overall Description

This project offers a comprehensive file management solution by allowing multiple users to perform various operations on files concurrently while maintaining data integrity and consistency. It supports simultaneous file reading by multiple threads, ensuring efficient access to shared

data. Exclusive write operations are implemented to prevent data corruption during concurrent modifications. Users can also delete or rename files, copy files to create duplicates, and view detailed metadata such as file size, creation date, and permissions. Additional features include file compression and decompression to optimize storage and robust error handling to ensure system reliability. A logging mechanism tracks all actions performed, aiding in auditing and troubleshooting.

## 5.1 Operating Environment

The Project needs a **Linux OS + tkinter** to be setup which can be installed from the internet.

## 6 Functional requirements

1. **Concurrent File Reading:** Enable multiple threads to read from a file simultaneously without conflicts.
2. **Exclusive File Writing:** Allow only one thread to write to a file at any given time.
3. **File Deletion:** Permit users to delete files securely through a thread-safe mechanism.
4. **File Renaming:** Provide a safe method for renaming files to prevent conflicts during operation.
5. **File Copying:** Support creating copies of files using dedicated threads.
6. **File Metadata Display:** Show file properties such as size, creation date, and permissions in the GUI.
7. **Error Handling:** Handle errors gracefully for operations like file not found, permission denied, or disk space issues.
8. **Logging Operations:** Maintain a log of all file operations performed for transparency and debugging.
9. **Compression and Decompression:** Provide file compression and decompression functionality for storage efficiency.

## 7 Non-Functional requirements

1. **Performance:** Ensure fast and efficient handling of file operations, even with multiple concurrent users.
2. **Scalability:** Support increasing numbers of concurrent users and files without significant performance degradation.
3. **Security:** Protect file access with proper locking mechanisms to prevent unauthorized or conflicting operations.
4. **Usability:** Offer a user-friendly GUI with intuitive controls for all supported operations.
5. **Reliability:** Guarantee stable operation with minimal downtime, even under heavy load or unexpected failures.
6. **Portability:** Ensure compatibility across different platforms for the client-side GUI.
7. **Maintainability:** Use a modular code structure to simplify future updates and enhancements.

## 8 System Constraints

1. The server must run on a system capable of supporting multithreading in C, with sufficient resources to handle concurrent operations.
2. The client GUI requires Python and its dependencies, including socket programming libraries, to function correctly.
3. File operations are limited to the permissions and storage capacity of the server's file system.
4. Network latency may impact communication speed between the client and server.
5. The system requires reliable network connectivity to ensure uninterrupted client-server interaction.
6. Compression and decompression operations may increase CPU usage during execution.

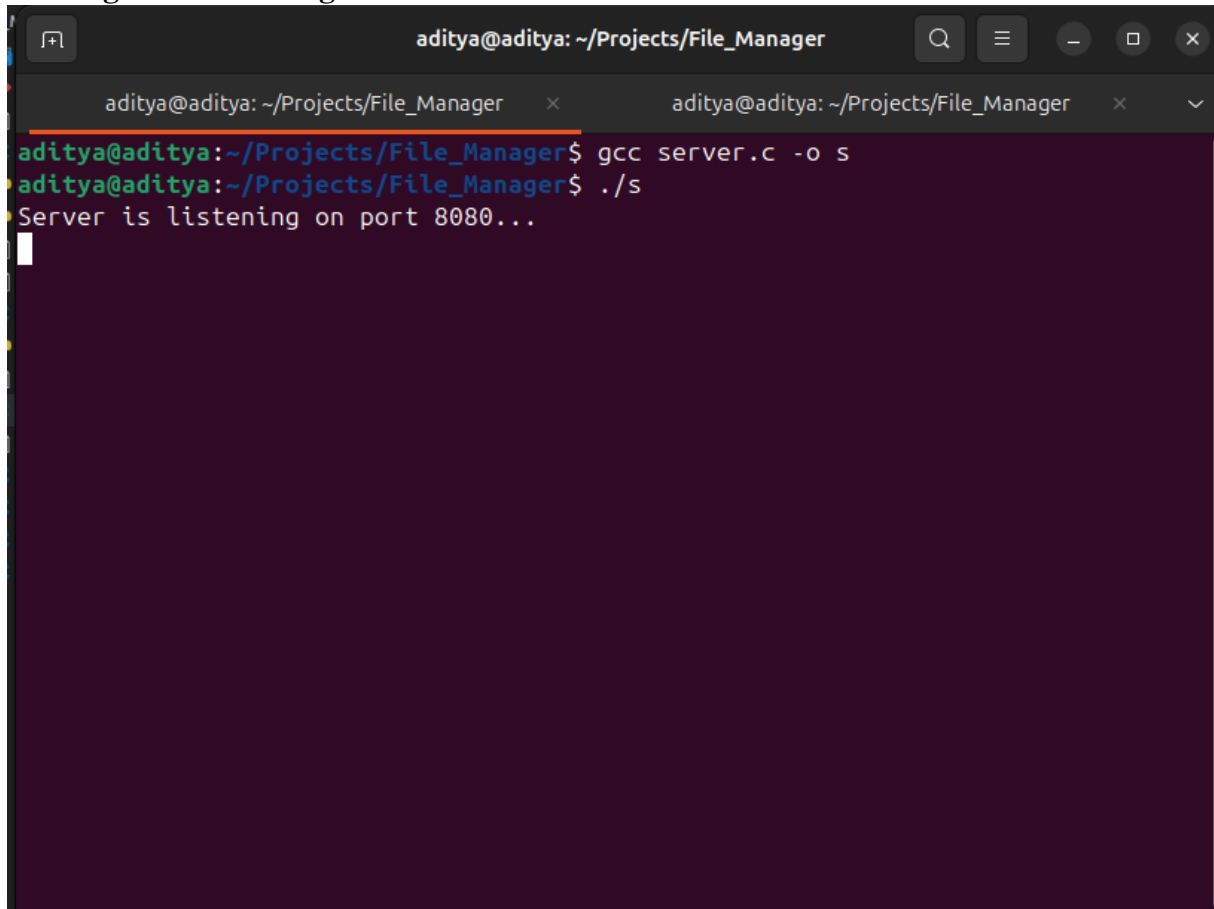
System Constraints

## **9 Assumptions and Dependencies**

One assumption about the product is that it will always be used on a device that has enough performance. If the system does not have enough hardware resources available for the application, for example the users might have allocated them with other applications, there may be scenarios where the application does not work as intended. Assumptions and Dependencies

## 10 Output Snippets

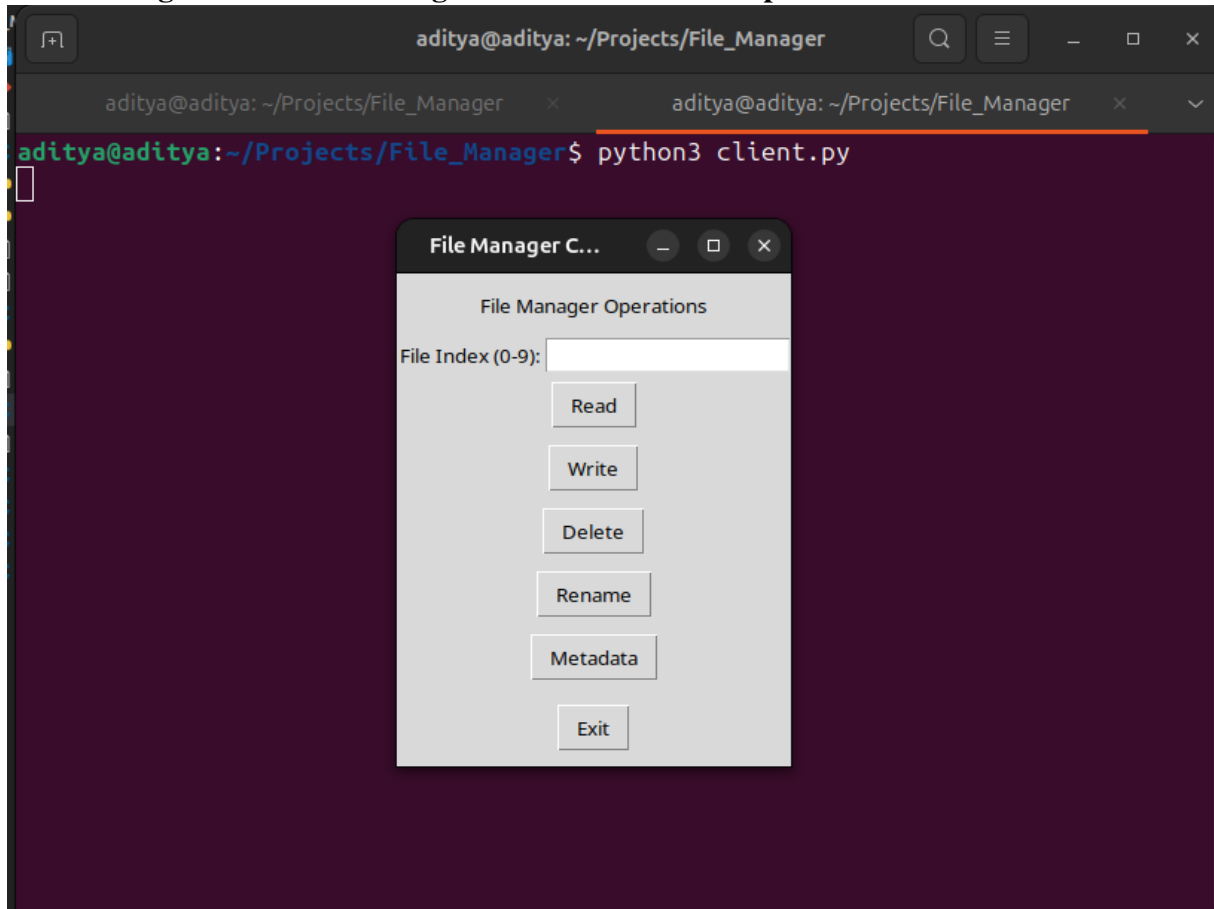
### Starting the File-Manager at PORT 8080

A terminal window with a dark purple background. The title bar shows 'aditya@aditya: ~/Projects/File\_Manager'. There are two tabs open, both with the same title. The terminal shows the following commands and output:

```
aditya@aditya:~/Projects/File_Manager$ gcc server.c -o s
aditya@aditya:~/Projects/File_Manager$ ./s
Server is listening on port 8080...
```

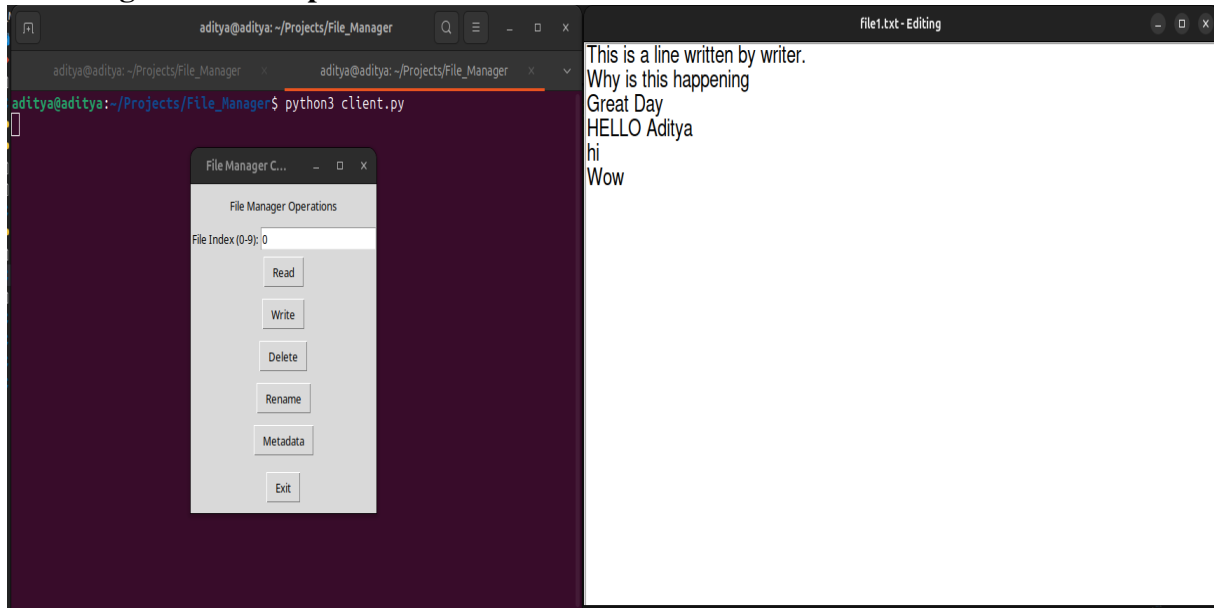
A white cursor is visible on the line following the output.

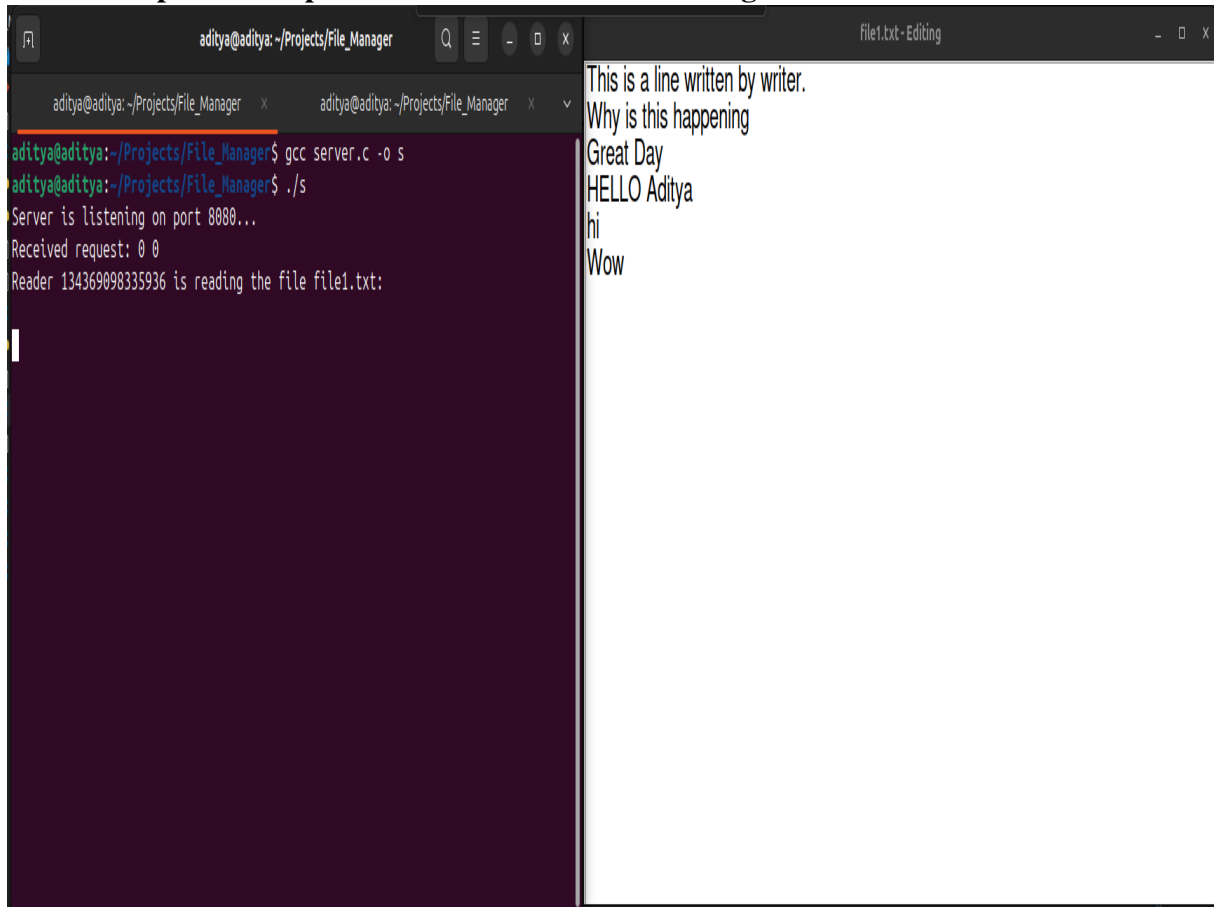
### Creating A new client analogous to new tab in file explorer





## Doing The Read Operation



**Read Operation Opens a custom reader made using tkinter**

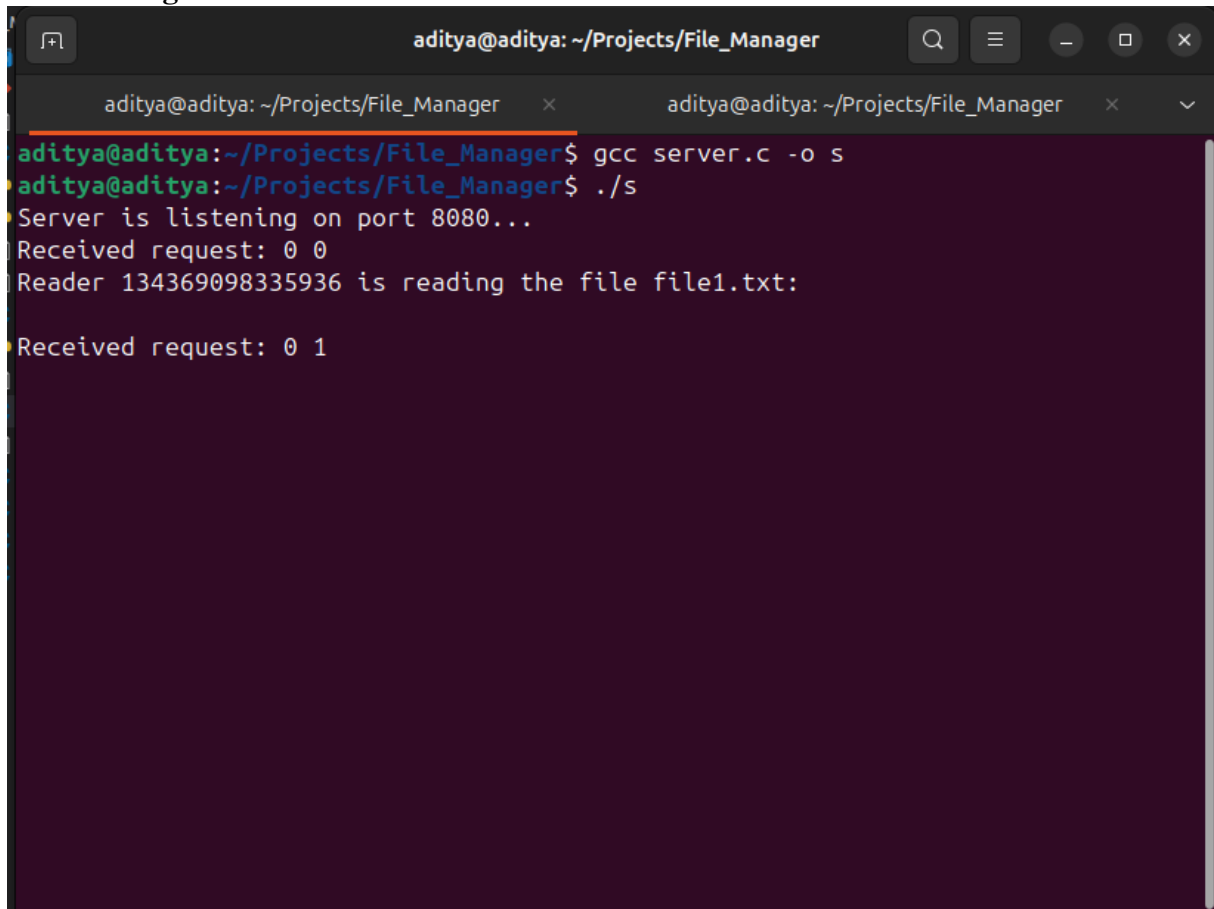
The screenshot shows a terminal window on the left and a text editor on the right. The terminal window has a title bar with the text 'aditya@aditya: ~/Projects/File\_Manager'. It contains the following text:

```
aditya@aditya:~/Projects/File_Manager$ gcc server.c -o s
aditya@aditya:~/Projects/File_Manager$ ./s
Server is listening on port 8080...
Received request: 0 0
Reader 134369098335936 is reading the file file1.txt:
```

The text editor window has a title bar with the text 'file1.txt - Editing'. It contains the following text:

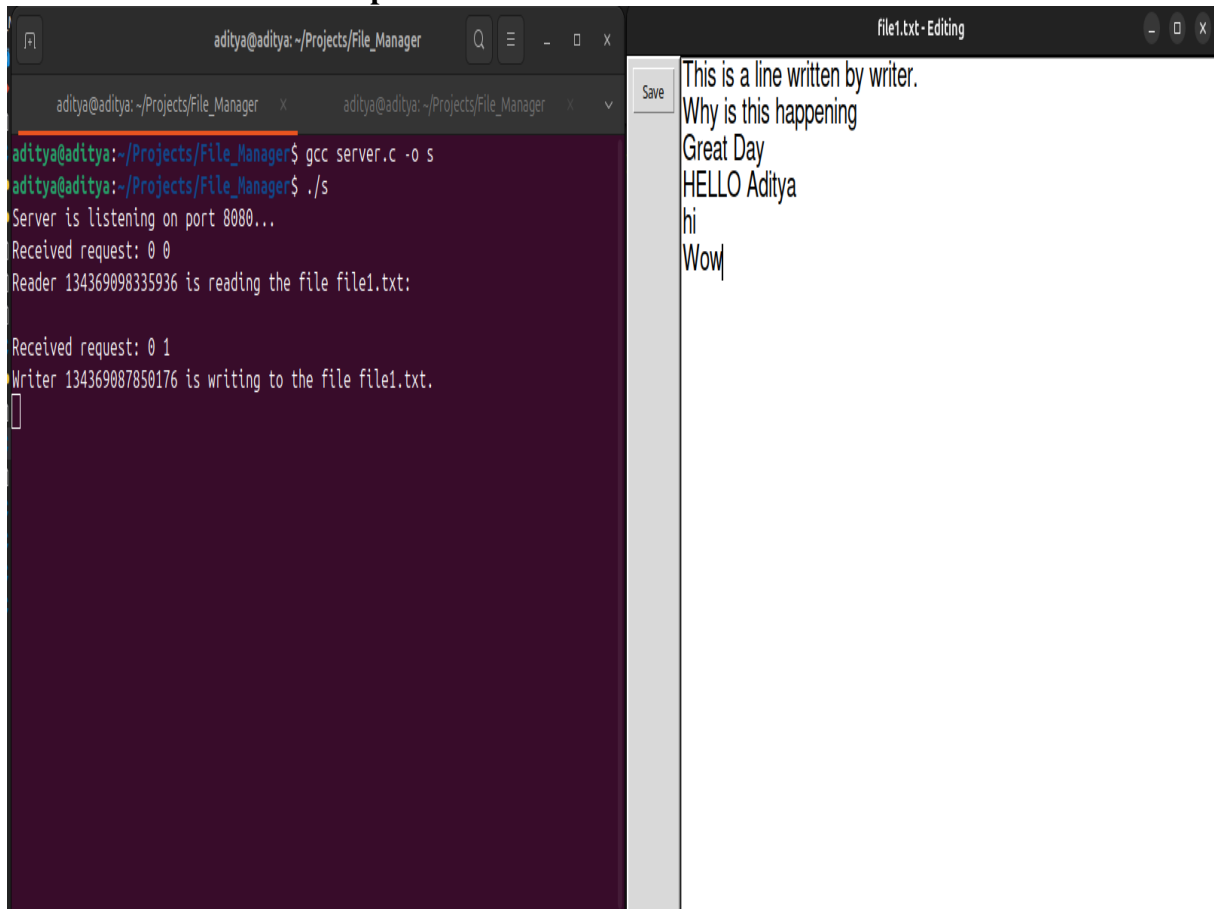
```
This is a line written by writer.
Why is this happening
Great Day
HELLO Aditya
hi
Wow
```

**While Reading** The server receives a write request but since there is a readlock it wont start writing.

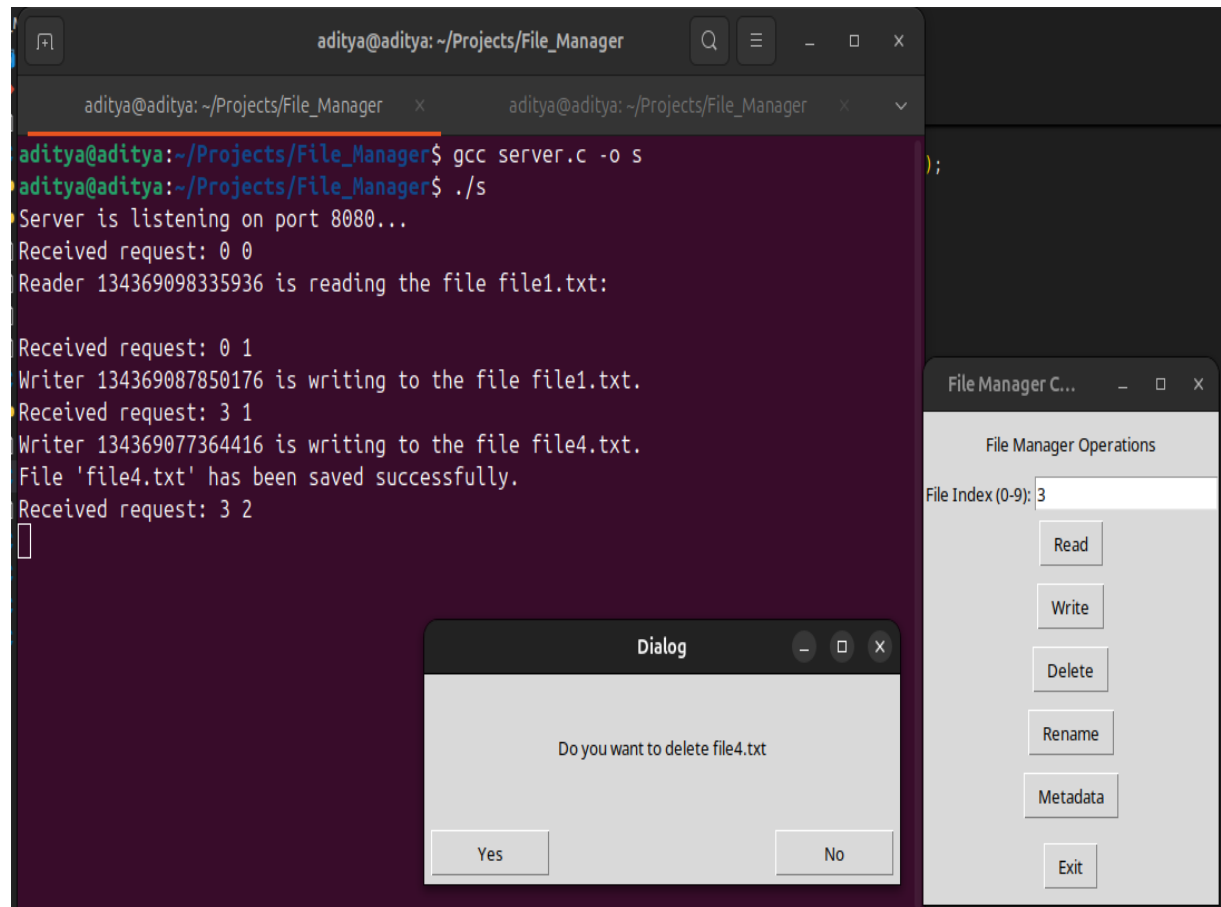
A terminal window titled 'aditya@aditya: ~/Projects/File\_Manager' with two tabs. The active tab shows the following commands and output:

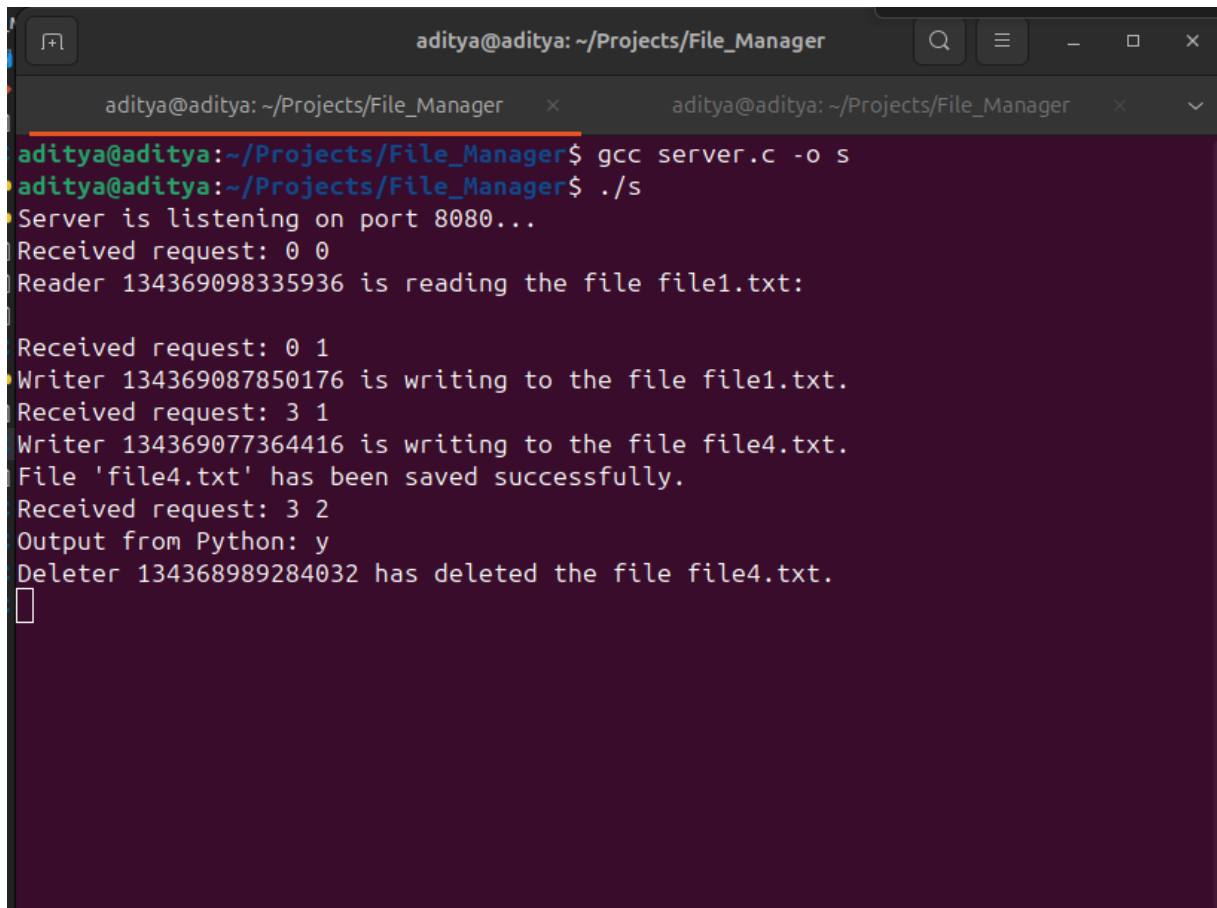
```
aditya@aditya:~/Projects/File_Manager$ gcc server.c -o s
aditya@aditya:~/Projects/File_Manager$ ./s
Server is listening on port 8080...
Received request: 0 0
Reader 134369098335936 is reading the file file1.txt:
Received request: 0 1
```

After we close the reading terminal the writer operation starts which also uses a custom **tkinter** window with save option



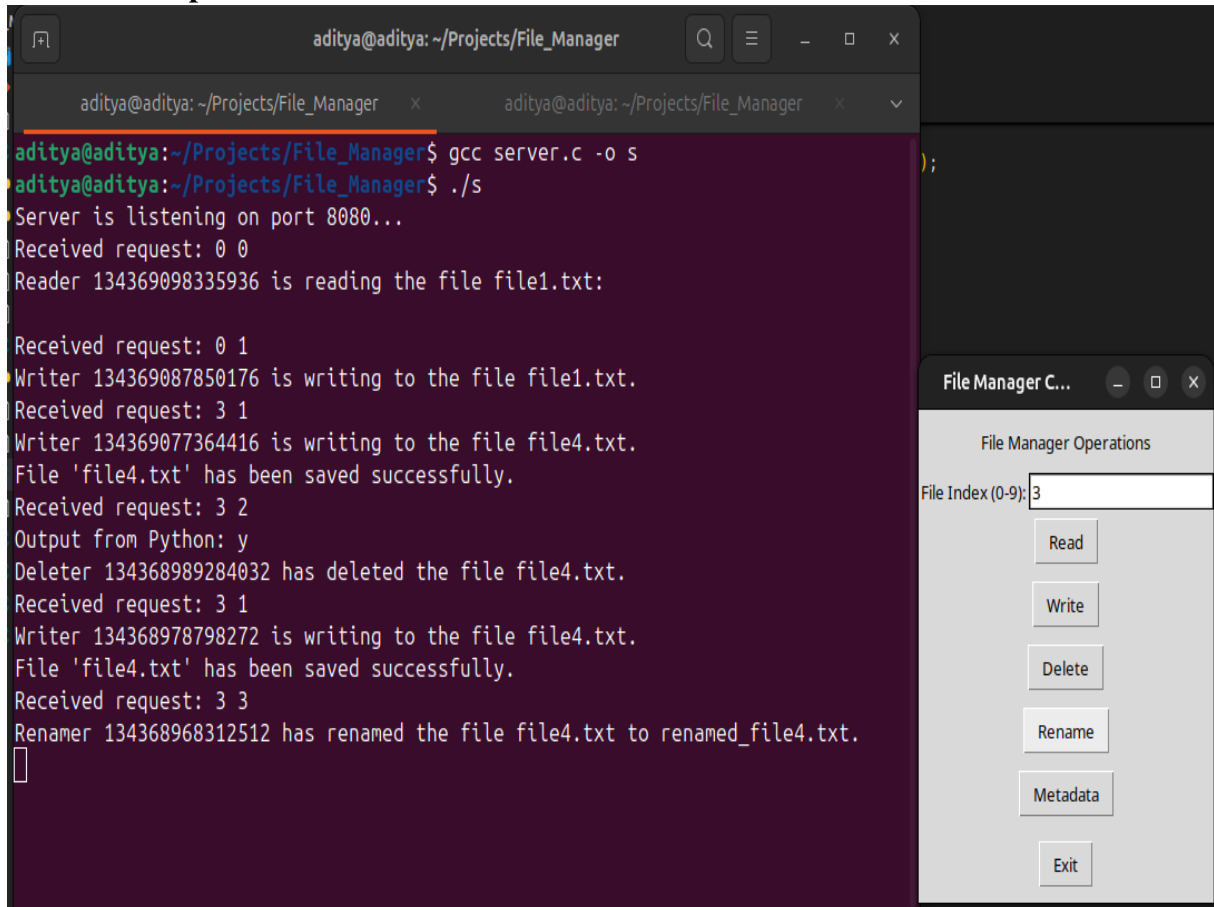
The delete option will open a dialog which asks confirmation to delete





```
aditya@aditya: ~/Projects/File_Manager
aditya@aditya: ~/Projects/File_Manager$ gcc server.c -o s
aditya@aditya: ~/Projects/File_Manager$ ./s
Server is listening on port 8080...
Received request: 0 0
Reader 134369098335936 is reading the file file1.txt:

Received request: 0 1
Writer 134369087850176 is writing to the file file1.txt.
Received request: 3 1
Writer 134369077364416 is writing to the file file4.txt.
File 'file4.txt' has been saved successfully.
Received request: 3 2
Output from Python: y
Deleter 134368989284032 has deleted the file file4.txt.
█
```

**Rename option renames the file to renamedfile.txt**

The screenshot displays a terminal window and a File Manager GUI. The terminal window, titled 'aditya@aditya: ~/Projects/File\_Manager', shows the execution of a C server program. The server listens on port 8080 and processes several requests: a read request for 'file1.txt', two write requests for 'file1.txt' and 'file4.txt', a delete request for 'file4.txt', and a rename request for 'file4.txt' to 'renamed\_file4.txt'. The File Manager GUI, titled 'File Manager C...', shows a 'File Index (0-9):' field with the value '3'. Below this field are buttons for 'Read', 'Write', 'Delete', 'Rename', 'Metadata', and 'Exit'.

```
aditya@aditya:~/Projects/File_Manager$ gcc server.c -o s
aditya@aditya:~/Projects/File_Manager$ ./s
Server is listening on port 8080...
Received request: 0 0
Reader 134369098335936 is reading the file file1.txt:

Received request: 0 1
Writer 134369087850176 is writing to the file file1.txt.
Received request: 3 1
Writer 134369077364416 is writing to the file file4.txt.
File 'file4.txt' has been saved successfully.
Received request: 3 2
Output from Python: y
Deleter 134368989284032 has deleted the file file4.txt.
Received request: 3 1
Writer 134368978798272 is writing to the file file4.txt.
File 'file4.txt' has been saved successfully.
Received request: 3 3
Renamer 134368968312512 has renamed the file file4.txt to renamed_file4.txt.
█
```

File Manager C... [min] [max] [close]

File Manager Operations

File Index (0-9): 3

Read

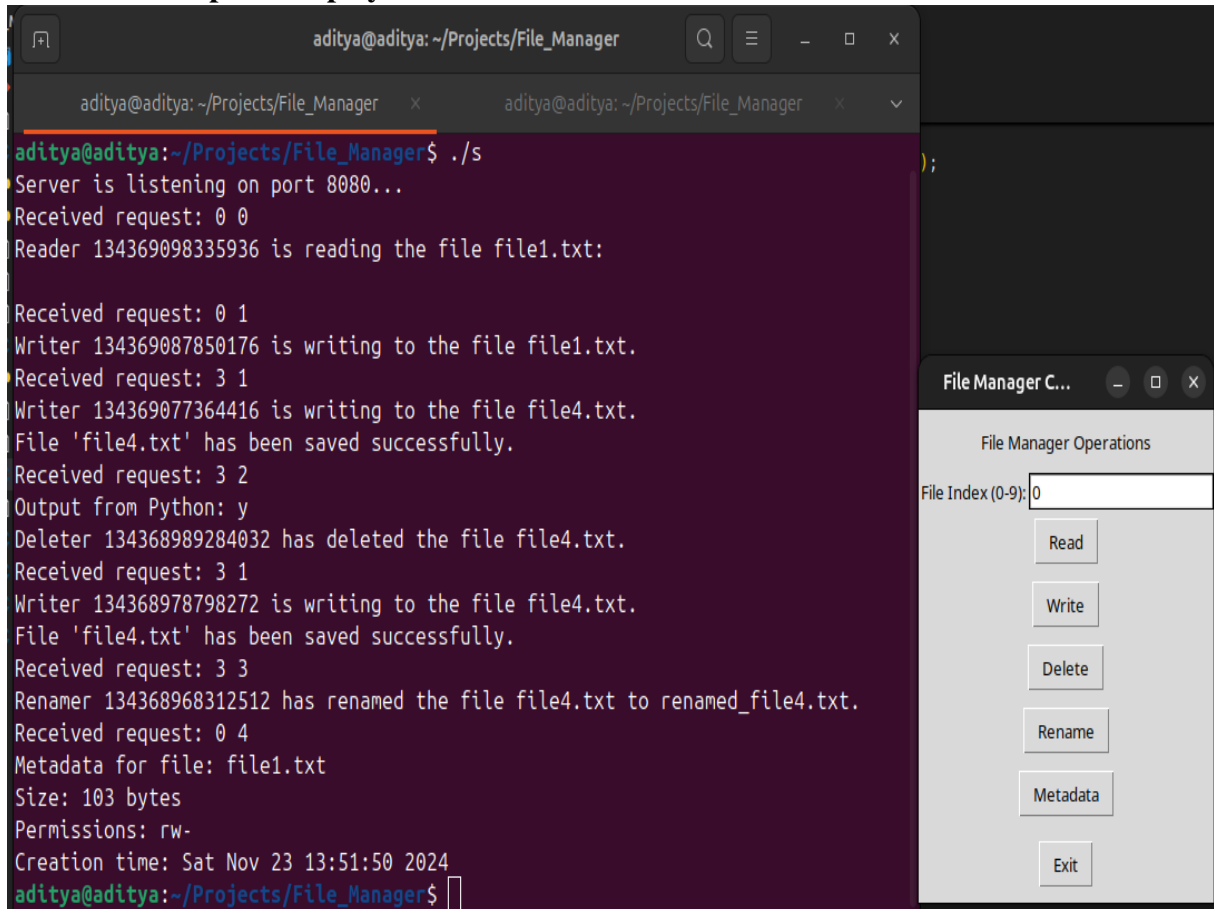
Write

Delete

Rename

Metadata

Exit

**Metadata option displays the file metadata on terminal**

```
aditya@aditya: ~/Projects/File_Manager
aditya@aditya: ~/Projects/File_Manager
aditya@aditya: ~/Projects/File_Manager$ ./s
Server is listening on port 8080...
Received request: 0 0
Reader 134369098335936 is reading the file file1.txt:

Received request: 0 1
Writer 134369087850176 is writing to the file file1.txt.
Received request: 3 1
Writer 134369077364416 is writing to the file file4.txt.
File 'file4.txt' has been saved successfully.
Received request: 3 2
Output from Python: y
Deleter 134368989284032 has deleted the file file4.txt.
Received request: 3 1
Writer 134368978798272 is writing to the file file4.txt.
File 'file4.txt' has been saved successfully.
Received request: 3 3
Renamer 134368968312512 has renamed the file file4.txt to renamed_file4.txt.
Received request: 0 4
Metadata for file: file1.txt
Size: 103 bytes
Permissions: rw-
Creation time: Sat Nov 23 13:51:50 2024
aditya@aditya: ~/Projects/File_Manager$
```

File Manager C...

File Manager Operations

File Index (0-9): 0

Read

Write

Delete

Rename

Metadata

Exit



```
Metadata for file: file1.txt
```

```
Size: 103 bytes
```

```
Permissions: rw-
```

```
Creation time: Sat Nov 23 13:51:50 2024
```

```
aditya@aditya:~/Projects/File_Manager$
```

**Any operation performed that can change file state is logged in the log file**

```
1 [1732350659.557541] Read from file: file1.txt
2 [1732350700.382897] Write from file: file1.txt
3 [1732350738.767804] Write from file: file4.txt
4 [1732350752.905061] Delete from file: file4.txt
5 [1732350815.178542] Write from file: file4.txt
6 [1732350819.559845] Rename from file: renamed_file4.txt
7
```

## 11 Conclusion

The Multithreaded File Management System is a robust and efficient solution for handling file operations in a concurrent environment. By leveraging the power of multithreading and a client-server architecture, the system ensures secure, reliable, and scalable management of files. Its comprehensive features, including concurrent reading, exclusive writing, file metadata display, and compression capabilities, make it a versatile tool for diverse file-handling needs. With robust error handling, user-friendly GUI, and logging for auditing, the system is designed to perform effectively in real-world scenarios, ensuring data integrity and operational efficiency. This project demonstrates the seamless integration of multithreading, socket programming, and cross-platform capabilities, providing a valuable resource for managing files in complex environments.