CSE 344

SYTEM PROGRAMING

HOMEWORK 1 REPORT

Student Name: Abdullah TÜRKMEN

Student ID: 210104004072

Introduction

In this homework, I created simple file manager utility written in C that provides basic file and directory operations from the command line. Here's what it does:

1. It allows users to:
    a. Create directories (createDir)
    b. Create files (createFile)
    c. List files in a directory (listDir)
    d. List files with a specific extension (listFilesByExtension)
    e. Read file contents (readFile)
    f. Append content to files (appendToFile)
    g. Delete files (deleteFile)
    h. Delete empty directories (deleteDir)
    i. Display operation logs (showLogs)
2. The program logs all successful operations to a log file called "log.txt" with timestamps.
3. It includes error handling for common file operation issues like permission denied, file not found, or directory not empty.
4. When run without arguments, it displays a help message with usage instructions for all the available commands.

Program code uses low-level system calls like open(), read(), write(), and fork() for file operations rather than higher-level C standard library functions.

Code Explanation

```
int main(int argc, char const *argv[]);
```

The main function parses command-line arguments and directs program flow:

- With no arguments, it displays usage instructions
- With arguments, it matches the first argument to a command name and calls the appropriate function with the remaining arguments
- It checks for the correct number of arguments for each command

```
int createDir(const char *folderName):
```

CreateDir function creates dir with spesific name.

- Checks if the folder name is null
- Uses stat to check if the directory already exists
- Calls mkdir to create the directory with permissions 0777
- Logs success or returns an error

```
int createFile(const char *fileName);
```

CreateFile function creates file with spesific name.

- Uses open with O_CREAT and O_EXCL flags to create a new file
- Handles common errors: file exists, permission denied
- Writes the current timestamp to the new file
- Logs the successful creation

```
int listDir(const char *folderName);
```

ListDir function acts like ls command. It list all file and directory in spesified folder.

- Creates a child process using fork

In the child process:

- Opens the directory using opendir
- Reads and displays each entry using readdir
- Logs the operation

The parent process waits for the child to complete.

```
int listFilesByExtension(const char *folderName, const char *extension);
```

Similar to listDir but filters entries by checking if they contain the specified extension

- Only displays files that match the extension

```
int readFile(const char *fileName);
```

ReadFile function reads file with spesific name.

- Opens the file in read-only mode
- Reads up to 1024 bytes into a buffer
- Writes the buffer contents to standard output
- Logs the read operation

```
int appendToFile(const char *fileName,const char *content);
```

AppendToFile function appends content to spesific file.

- Opens the file in append mode
- Uses flock to lock the file during writing
- Appends the provided content
- Unlocks and closes the file
- Logs the append operation

```
int deleteFile(const char *fileName);
```

DeleteFile deletes file with spesific name.

- Forks a child process
- Child process calls unlink to delete the file
- Handles errors like permission denied or file not found
- Logs successful deletion

```
int deleteDir(const char *folderName);
```

DeleteDir deletes directory with spesific directory in current directory

- Forks a child process
- Child process calls rmdir to delete the directory
- Handles errors including "directory not empty"
- Logs successful deletion

```
void log_message(const char *message);
```

Log_mesage logs message to log.txt if you want to change log file, you have to change open function parameter.

- Opens log.txt in append mode (creating it if needed)
- Gets the current time and formats it
- Writes a formatted log entry with timestamp and message
- Each log entry has a timestamp, dash, message, and newline

```
int showLogs();
```

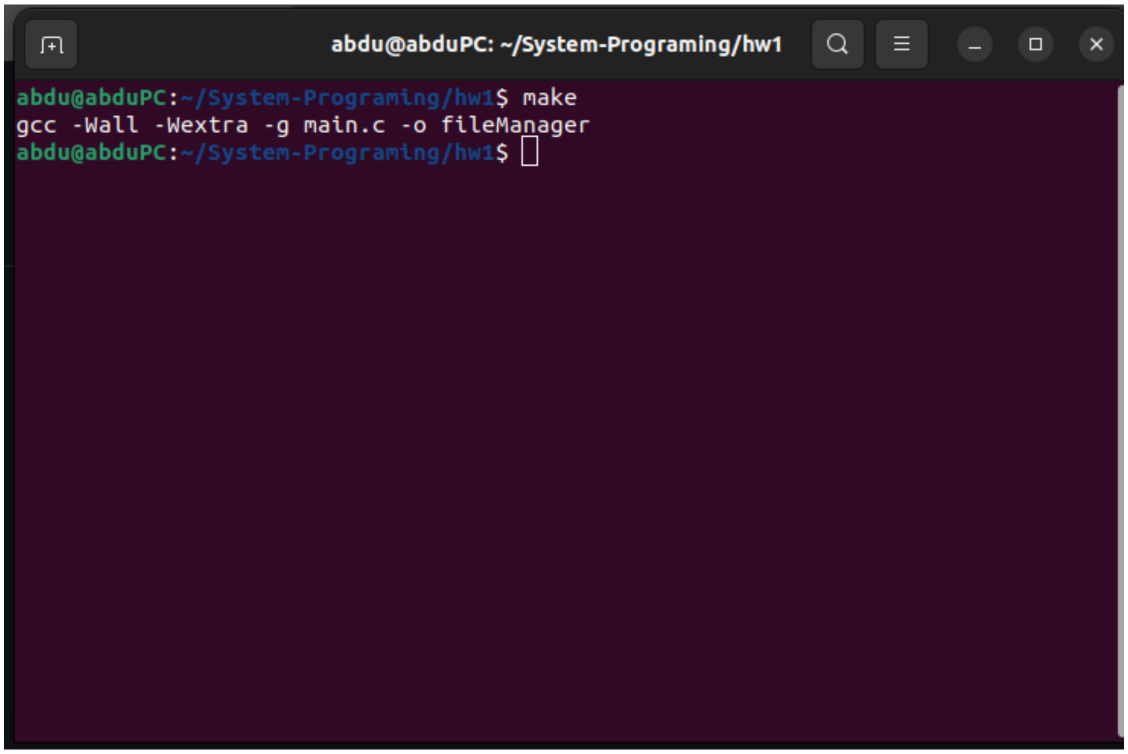
ShowLogs reads log.txt and prints to stdout.

- Opens log.txt in read-only mode

- Reads and displays up to 1024 bytes of the log content
- Logs that the logs were displayed

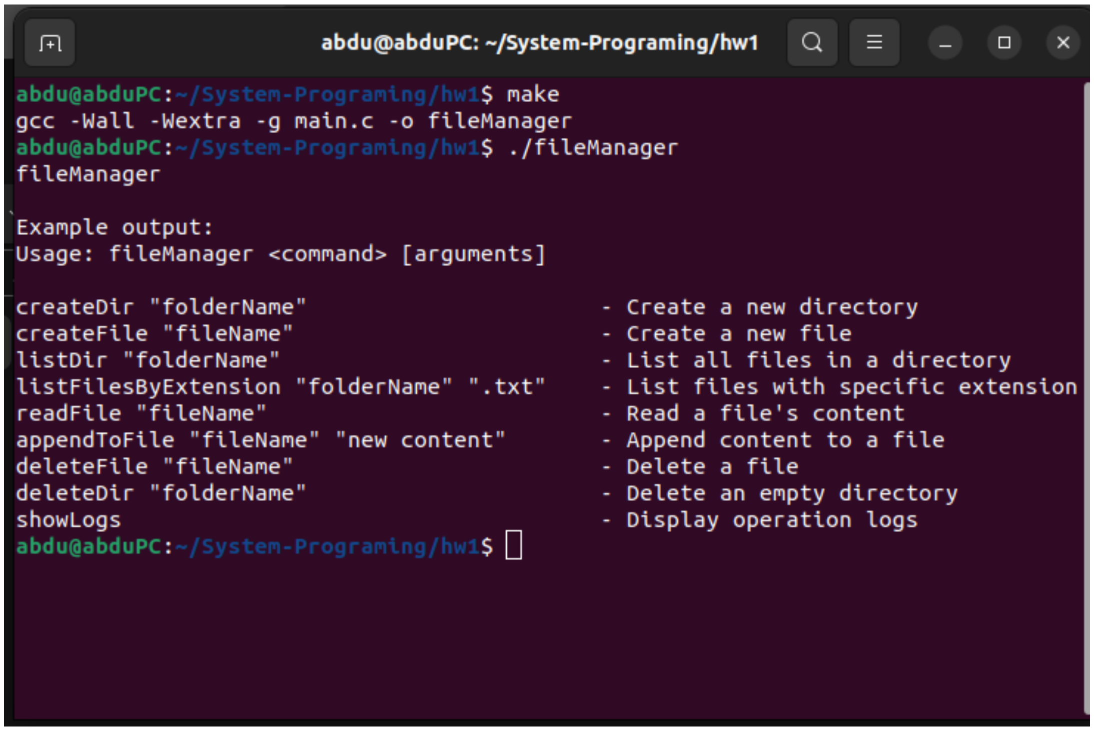
ScreenShots of Homeworks

Make :



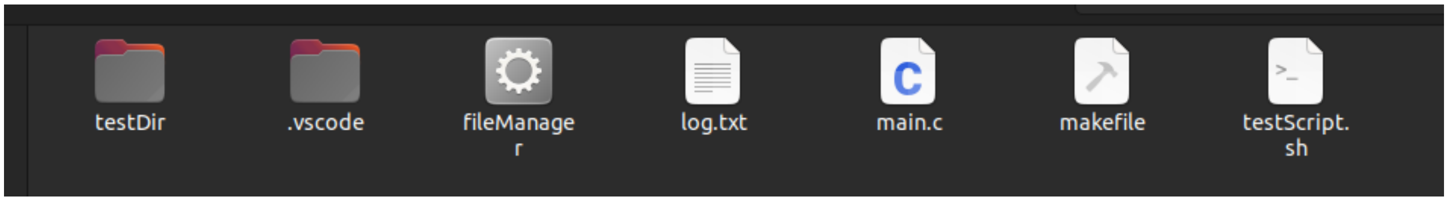
```
abdu@abduPC:~/System-Programing/hw1$ make
gcc -Wall -Wextra -g main.c -o fileManager
abdu@abduPC:~/System-Programing/hw1$ 
```

./fileManager :

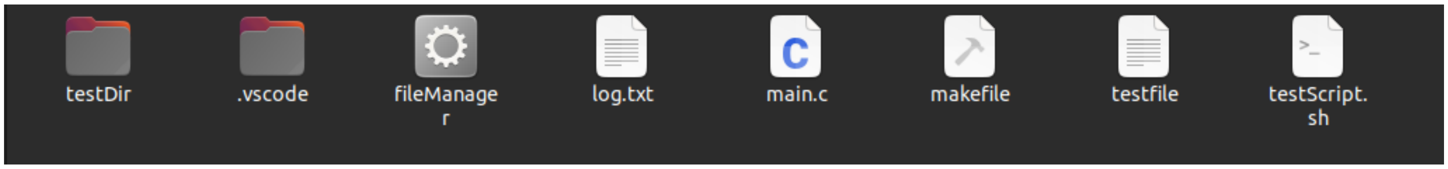


```
abdu@abduPC:~/System-Programing/hw1$ make
gcc -Wall -Wextra -g main.c -o fileManager
abdu@abduPC:~/System-Programing/hw1$ ./fileManager
fileManager

Example output:
Usage: fileManager <command> [arguments]

createDir "folderName"                       - Create a new directory
createFile "fileName"                        - Create a new file
listDir "folderName"                         - List all files in a directory
listFilesByExtension "folderName" ".txt"     - List files with specific extension
readFile "fileName"                          - Read a file's content
appendToFile "fileName" "new content"        - Append content to a file
deleteFile "fileName"                        - Delete a file
deleteDir "folderName"                       - Delete an empty directory
showLogs                                     - Display operation logs
abdu@abduPC:~/System-Programing/hw1$ 
```

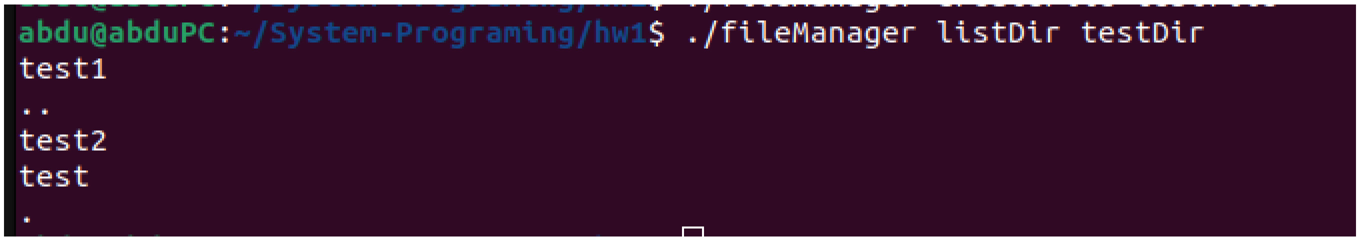
./fileManager createDir testdir



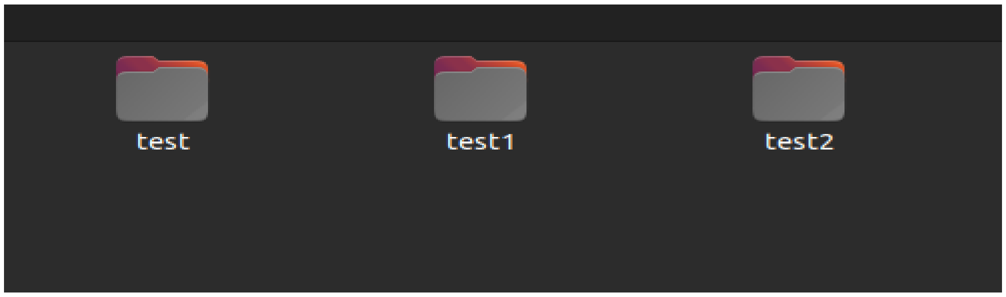
./fileManager createFile testfile



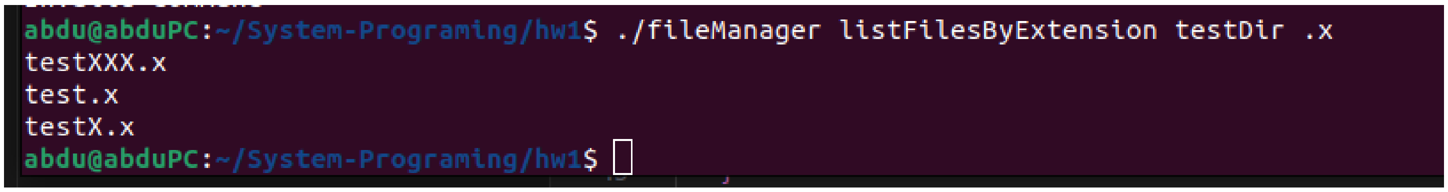
./fileManager listDir testDir



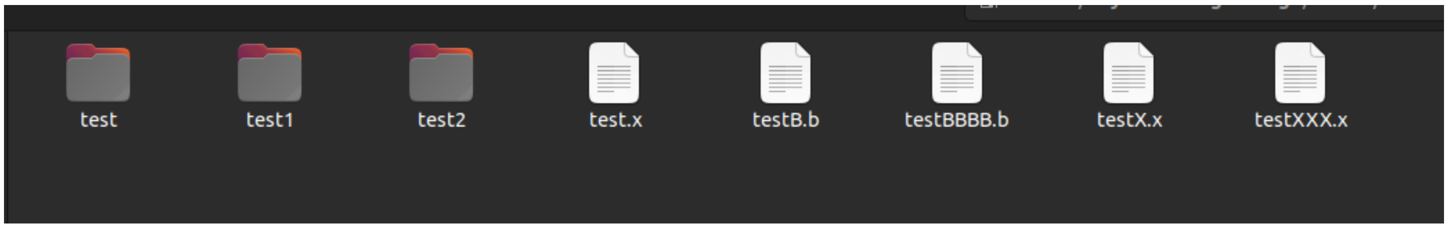
```
abdu@abduPC:~/System-Programing/hw1$ ./fileManager listDir testDir
test1
..
test2
test
.
```



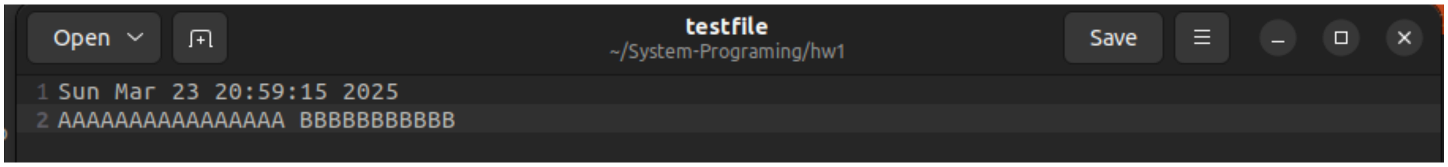
./fileManager listFileByExtension .x



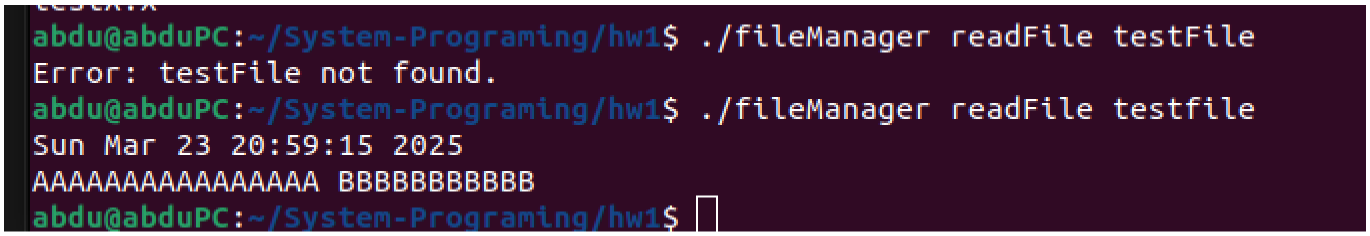
```
abdu@abduPC:~/System-Programing/hw1$ ./fileManager listFilesByExtension testDir .x
testXXX.x
test.x
testX.x
abdu@abduPC:~/System-Programing/hw1$
```



./fileManager readFile testFile

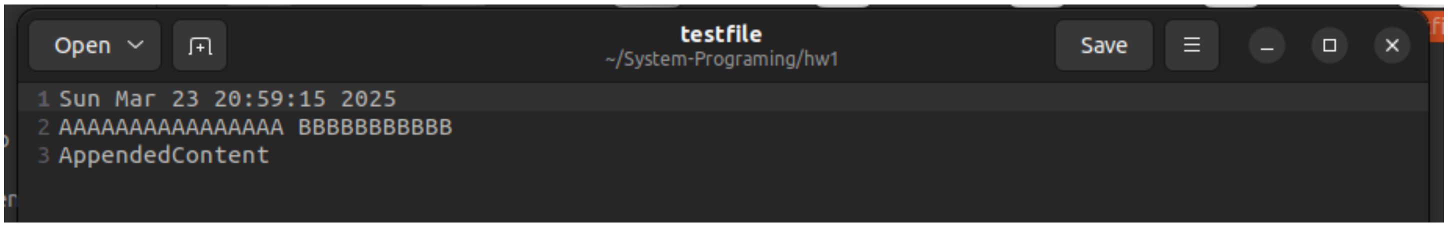


```
Sun Mar 23 20:59:15 2025
AAAAAAAAAAAAAAAA BBBBBBBBBBB
```



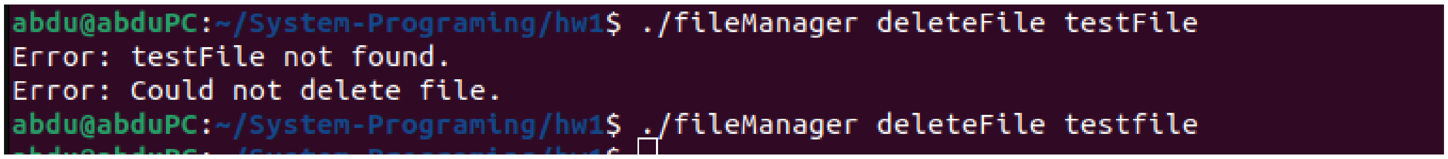
```
abdu@abduPC:~/System-Programing/hw1$ ./fileManager readFile testFile
Error: testFile not found.
abdu@abduPC:~/System-Programing/hw1$ ./fileManager readFile testfile
Sun Mar 23 20:59:15 2025
AAAAAAAAAAAAAAAA BBBBBBBBBBB
abdu@abduPC:~/System-Programing/hw1$
```

./fileManager appendToFile testfile AppendedContent

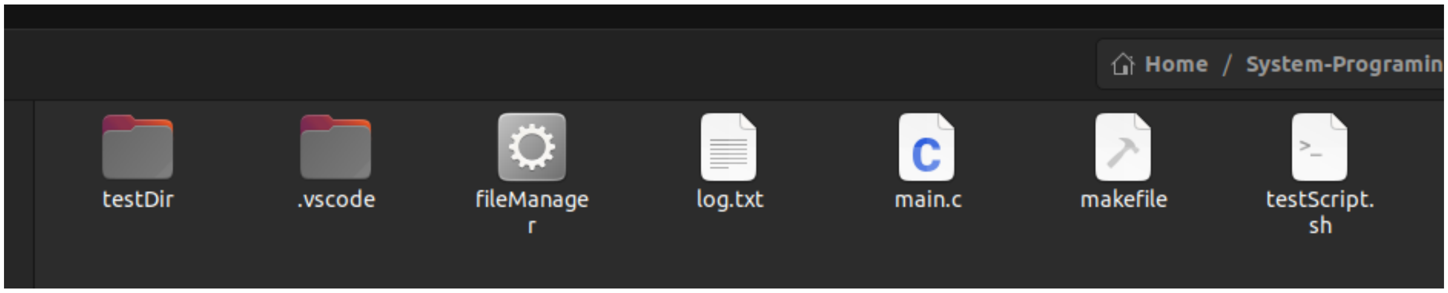


```
Sun Mar 23 20:59:15 2025
AAAAAAAAAAAAAAAA BBBBBBBBBBB
AppendedContent
```

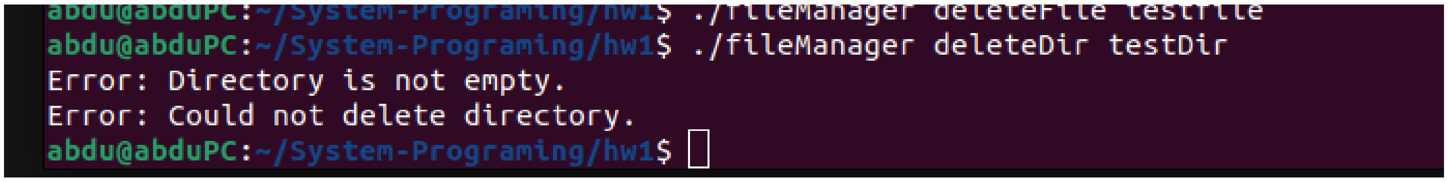
./fileManager deleteFile testFile



```
abdu@abduPC:~/System-Programing/hw1$ ./fileManager deleteFile testFile
Error: testFile not found.
Error: Could not delete file.
abdu@abduPC:~/System-Programing/hw1$ ./fileManager deleteFile testfile
```

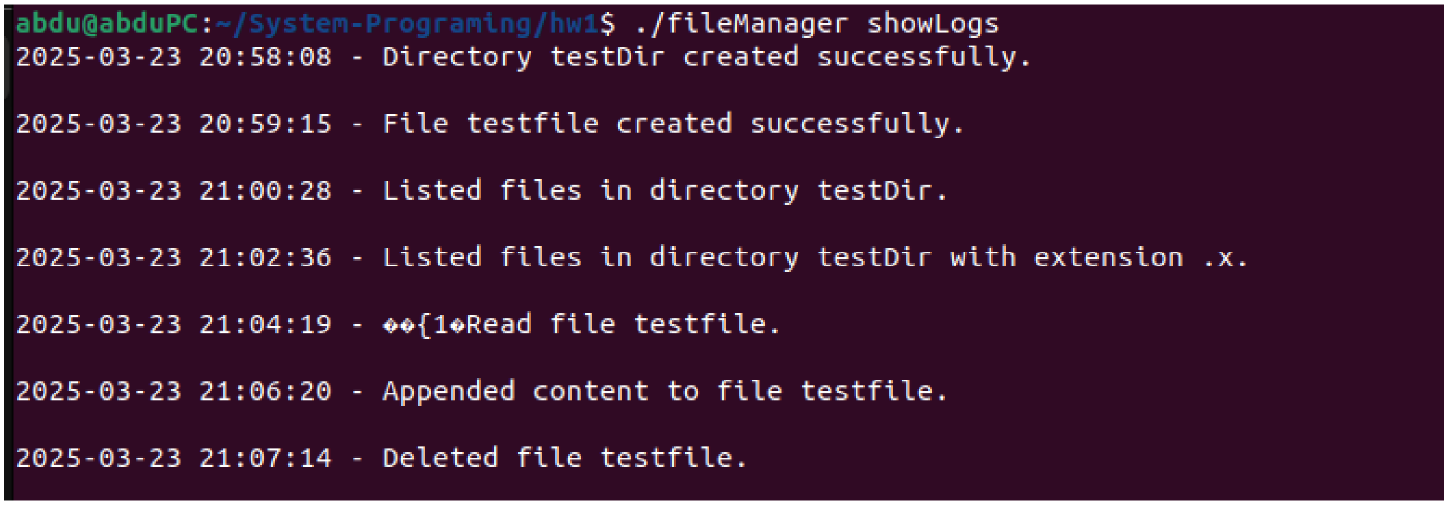


./fileManager DeleteDir testDir



```
abdu@abduPC:~/System-Programing/hw1$ ./fileManager deleteDir testDir
Error: Directory is not empty.
Error: Could not delete directory.
abdu@abduPC:~/System-Programing/hw1$
```

./fileManager showLogs



```
abdu@abduPC:~/System-Programing/hw1$ ./fileManager showLogs
2025-03-23 20:58:08 - Directory testDir created successfully.

2025-03-23 20:59:15 - File testfile created successfully.

2025-03-23 21:00:28 - Listed files in directory testDir.

2025-03-23 21:02:36 - Listed files in directory testDir with extension .x.

2025-03-23 21:04:19 - ��{1�Read file testfile.

2025-03-23 21:06:20 - Appended content to file testfile.

2025-03-23 21:07:14 - Deleted file testfile.
```

Conclusion: Challenges Faced, Solutions, and Final Thoughts

Development Challenges

When I built this file manager, I ran into several challenges:

I had to use basic system functions like open(), read(), and write() instead of easier C functions. This made everything harder because I had to handle all the small details myself.

I also struggled with string handling in C. When I tried to combine strings in the logging functions, I kept running into memory problems because C doesn't check if strings get too long.

Using fork() to create child processes was tricky too. I had to make sure parent and child processes communicated properly and that no zombie processes were left behind.

Solutions I Found

To fix these problems:

I carefully checked all return values and used error codes to show helpful messages when things went wrong.

For string problems, I was more careful with buffer sizes and how I used functions like strcat(). I made sure my logging system used fixed-size buffers that wouldn't overflow.

For process management, I made child processes do the risky operations while parent processes waited for them to finish. This kept the program more stable.

I also used file locking with flock() when adding text to files so that multiple users couldn't mess up the files by writing at the same time.

Final Thoughts

Building this file manager taught me a lot about how computers handle files at a basic level. Working with system calls directly instead of using higher-level functions gave me a better understanding of what happens behind the scenes.

If I were to improve this program, I would add features like copying files, deleting folders with files in them, and changing file permissions. I would also make the logging system better at handling large log files.

The most valuable lesson was learning how file systems actually work. While using libraries would have been faster for development, I now understand the importance of proper error handling and resource management.