

**GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
DEPARTMENT**

**CSE344 Systems Programming
Spring 2025
Homework 1 Report**

**Osmancan Bozali
220104004011**

Introduction:

This report presents the implementation of a Secure File and Directory Management System developed in C for CSE344 Systems Programming course. The program utilizes Linux system calls to perform file and directory operations, including creation, listing, reading, updating, and deletion, while incorporating process creation with `fork()` and logging functionality. The system ensures secure file operations through file locking. This assignment was tested on Debian 11 (64-bit) in VirtualBox on my machine, adhering to the provided requirements. The purpose of this homework is to simulate a basic file management utility and operation logging.

Code Explanation:

The program has several functions, each is written to handle a specific file or directory operation (or helper function for operations). Below is an explanation of the functions and their roles (see the source code for full function implementations):

writeMessage(int fd, const char* prefix, const char* name, const char* suffix): This utility function writes a formatted message to a specified file descriptor. It surrounds the name parameter with quotes, ensuring consistent error message formatting (e.g., Error: File "example.txt" not found). It's used for user feedback and aligns with the assignment's output requirements.

getTimestamp(): Generates a timestamp in the format [YYYY-MM-DD HH:MM:SS] using `time()` and `strftime()`. It returns a string, which is used for logging operations and writing creation timestamps into new files.

logOperationToFile(const char* operation, const char* name, const char* result): Appends a log entry to `log.txt` with a timestamp, operation type, target name (if applicable), and result. It uses `open()` with `O_APPEND` to ensure logs are cumulative.

createDirectory(const char *name): Creates a directory using `mkdir()` with permissions `0777`. If the directory exists (`EEXIST`), it outputs an error and logs the failure. Otherwise, it logs success. This implements the `createDir` command with proper error handling.

createFile(const char *name): Creates a file using `open()` with `O_CREAT | O_EXCL` to prevent overwriting existing files. On success, it writes the current timestamp to the file. Errors (e.g., file exists) trigger appropriate messages and logs.

listDirectory(const char *name): Uses `fork()` to create a child process that lists all files in the directory with `opendir()` and `readdir()`. The child exits with status 0 on success or 1 if the directory isn't found, and the parent logs the outcome.

listFilesByExtension(const char *name, const char *extension): Similar to `listDirectory()`, but filters files by extension using `strstr()`. It uses `fork()` and counts matching files; if none are found, it prints a specific message (e.g., No file with extension ".txt" found in "testDir"). The operation is logged at the end.

readFile(const char *name): Opens a file with `O_RDONLY` and reads its content into a buffer, printing it to stdout using `write()`. If the file doesn't exist, it outputs an error and logs the failure.

appendToFile(const char *name, const char *content): Opens a file with `O_APPEND` and uses `flock(LOCK_EX | LOCK_NB)` to lock it exclusively, preventing simultaneous writes. If locked or inaccessible, it reports an error; otherwise, it appends the content and logs success.

deleteFile(const char *name): Uses `fork()` to delete a file with `unlink()` in a child process. The parent waits and logs success or failure (e.g., file not found). This implements `deleteFile` with process separation as required.

deleteDirectory(const char *name): Uses `fork()` to delete an empty directory with `rmdir()` in a child process. It handles errors like non-empty directories (`ENOTEMPTY`) or non-existent directories (`ENOENT`), logging the result.

showLogs():

Reads and displays the contents of `log.txt` using `read()` and `write()`. If the log file is missing, it reports an error. This operation is logged in the `log.txt` too.

showHelp():

Prints a usage guide with all commands and descriptions when no arguments are provided (e.g., Usage: `fileManager <command> [arguments]`). It logs the display action.

main(int argc, char *argv[]):

The entry point parses command-line arguments and calls the appropriate function based on the command (e.g., `createDir`, `showLogs`). It checks for missing arguments and triggers `showHelp()` if no command is given, ensuring robust command handling.

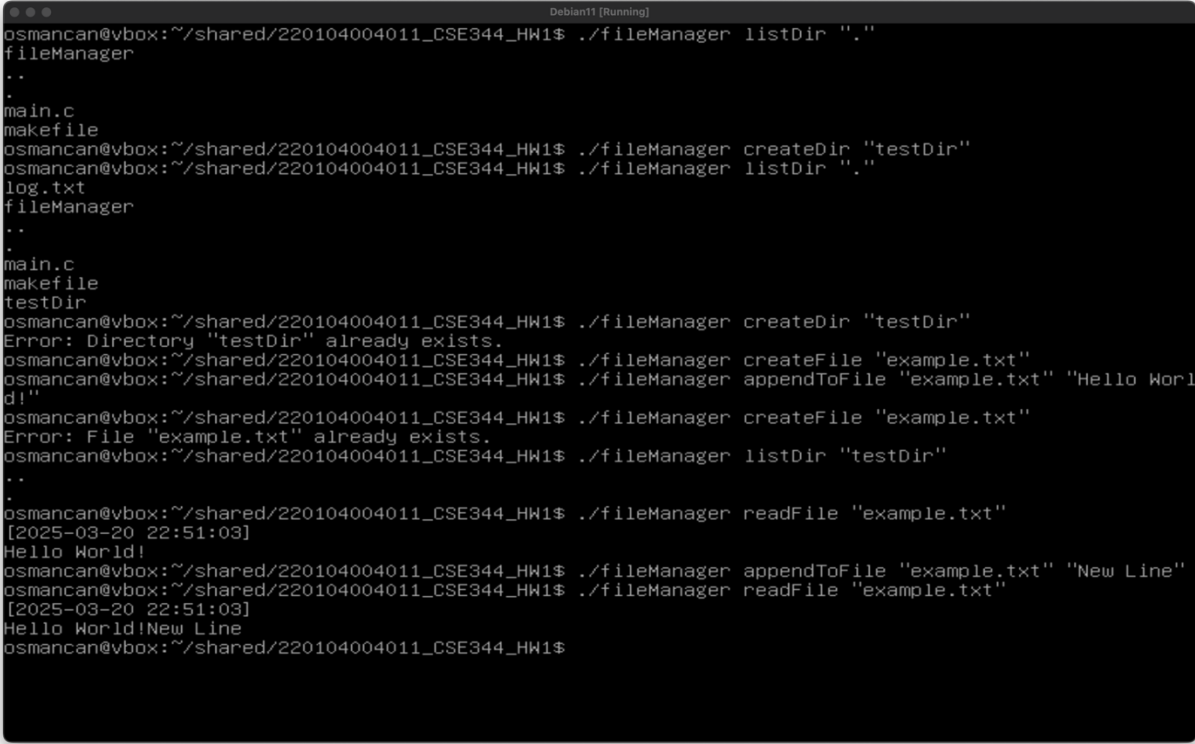
Each function uses system calls like `open()`, `write()`, and `fork()` to interact with the file system securely and logs operations consistently, adhering to the assignment's specifications.

Tests and Screenshots:

In order to test the program yourself, you can use the makefile

- make : This command cleans the all related files/folders and compiles the program (preferred, just run this command and everything will be ready)
- make clean : This command clears all the files/folders created while running the program)

listDir, createDir, createFile, appendToFile, readFile function tests with edge cases:



```
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listDir "."
fileManager
..
.
main.c
makefile
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager createDir "testDir"
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listDir "."
log.txt
fileManager
..
.
main.c
makefile
testDir
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager createDir "testDir"
Error: Directory "testDir" already exists.
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager createFile "example.txt"
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager appendToFile "example.txt" "Hello World!"
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager createFile "example.txt"
Error: File "example.txt" already exists.
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listDir "testDir"
..
.
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager readFile "example.txt"
[2025-03-20 22:51:03]
Hello World!
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager appendToFile "example.txt" "New Line"
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager readFile "example.txt"
[2025-03-20 22:51:03]
Hello World!New Line
osman@vbox:~/shared/220104004011_CSE344_HW1$
```

readFile, listFilesByExtension, deleteFile, listDir, deleteDir function tests with edge cases:

```
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager readFile "notexist.txt"
Error: File "notexist.txt" not found.
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listFilesByExtension "." ".txt"
log.txt
example.txt
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listFilesByExtension "." ".c"
main.c
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listFilesByExtension "testDir" ".txt"
No file with extension ".txt" found in "testDir".
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listFilesByExtension "notExistingDir" ".txt"
Error: Directory "notExistingDir" not found.
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager deleteFile "example.txt"
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listDir "."
log.txt
fileManager
..
.
main.c
makefile
testDir
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager deleteDir "testDir"
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listDir "."
log.txt
fileManager
..
.
main.c
makefile
osman@vbox:~/shared/220104004011_CSE344_HW1$
```

showLogs function test (logs of previous test results above):

```
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager showLogs
[2025-03-20 22:49:55] Directory "." listed successfully.
[2025-03-20 22:50:42] Directory "testDir" created successfully.
[2025-03-20 22:50:46] Directory "." listed successfully.
[2025-03-20 22:50:48] Directory "testDir" creation failed (already exists).
[2025-03-20 22:51:03] File "example.txt" created successfully.
[2025-03-20 22:51:25] Content appended to "example.txt" successfully.
[2025-03-20 22:51:39] File "example.txt" creation failed (already exists).
[2025-03-20 22:51:48] Directory "testDir" listed successfully.
[2025-03-20 22:52:00] File "example.txt" read successfully.
[2025-03-20 22:52:11] Content appended to "example.txt" successfully.
[2025-03-20 22:52:12] File "example.txt" read successfully.
[2025-03-20 22:52:53] File "notexist.txt" read failed. (file not found)
[2025-03-20 22:53:09] Directory "." listed by extension successfully.
[2025-03-20 22:53:13] Directory "." listed by extension successfully.
[2025-03-20 22:53:25] Directory "testDir" listed by extension successfully.
[2025-03-20 22:53:33] Directory "notExistingDir" listing by extension failed. (directory not found)
[2025-03-20 22:53:50] File "example.txt" deleted.
[2025-03-20 22:53:57] Directory "." listed successfully.
[2025-03-20 22:54:23] Directory "testDir" deleted.
[2025-03-20 22:54:25] Directory "." listed successfully.
osman@vbox:~/shared/220104004011_CSE344_HW1$ _
```

deleteDir with non-empty directory and showHelp function tests:

```
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager createDir "testDir"
osman@vbox:~/shared/220104004011_CSE344_HW1$ touch testDir/test.txt
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager listFilesByExtension "testDir" ".txt"
test.txt
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager deleteDir "testDir"
Error: Directory "testDir" is not empty.
osman@vbox:~/shared/220104004011_CSE344_HW1$ ./fileManager
Usage: fileManager <command> [arguments]
Commands:
  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
osman@vbox:~/shared/220104004011_CSE344_HW1$ _
```

Conclusion:

This homework was a great exercise in understanding Linux system calls, process management with `fork()`, and file locking with `flock()`. While implementing the homework I made research on the web consistently since I am a beginner in terms of Systems Programming. One challenge was ensuring proper error handling for all edge cases, such as existing files or locked resources, which I solved with `errno` and its error codes. Another difficulty was managing child processes effectively led me to use `waitpid()` to synchronize parent and child execution. Also the string formatting was a bit challenging. Overall, the implementation meets all requirements, and the testing scenario added as a proof. By implementing this homework, I think I made a good introduction to Systems Programming.