UNIVERSITI MALAYSIA SARAWAK

Faculty of Computer Science and Information Technology

Assignment/Project/Report Cover Sheet

Student Name	Student ID Number	Group	Signature
NOUF HASYA ILMA BINTI BADARUDDIN	78334	2	
NUR RAIYANI BINTI MOHD YUSRI AZHAR	80685	2	
ROFINNA ELLYA EMBANG ANAK UMAR @ RICHARD	81056	2	
SHEANA KASIH BENEDICT	78525	2	Sheanakb

Course Code: TMN 4133		**Course Name:** System Programming	
Assignment/Project/Report Title:	System Programming Project	**Lecturer:**	AP Dr Johari bin Abdullah
Due Date: 10th January 2025		**Date Submitted:** 10 January 2025	

This cover sheet must be completed, signed, and firmly attached to the front of the submission. All work must be submitted by the due date. If an extension of work is granted, an extension acknowledgement slip must be signed by the lecturer/tutor and attached to the assignment/project/report. Please note that it is your responsibility to retain copies of your assignment.

Plagiarism and Collusion are methods of cheating that are addressed in the Peraturan Akademik Universiti Malaysia Sarawak para 11: Etika Akademik

Plagiarism

Plagiarism is the presentation of work which has been copied in whole or in part from another person's work, or from any other source such as the Internet, published books or periodicals without due acknowledgement given in the text.

Collusion

Collusion is the presentation of work that is the result in whole or in part of unauthorized collaboration with another person or persons.

Where there are reasonable grounds for believing that cheating has occurred, the only action that may be taken when plagiarism or collusion is detected is for the staff member not to mark the item of work and to report or refer the matter to the Dean. This may result in work being disallowed and given a fail grade or if the circumstances warrant, the matter may be referred to a committee of inquiry for investigation. Such investigation may result in the matter being referred to the University Discipline Committee, which has the power to exclude a student.

Upon placing signature above, I certify that I have not plagiarized the work of others or participated in unauthorized collusion when preparing this assignment.

I also certify that I have taken proper case in safeguarding my work and have made all reasonable efforts to ensure that my work is not able to be copied.

MARK :

Table of Contents

1.0 INTRODUCTION 4

2.0 TASK A: CODING TASK 5

- 2.1 COMPILING AND EXECUTING SUPERCOMMAND.C 10
- 2.2 FILE OPERATIONS 11
  - 2.2.1 Create/Open a File using open() system call 14
  - 2.2.2 Deleting a File using unlink() system call 15
  - 2.2.3 Reading a File using read() system call 16
  - 2.2.4 Writing into a File using write () system call 16
  - 2.2.5 Changing File Permissions using chmod() system call 17
- 2.2 DIRECTORY OPERATIONS 18
  - 2.2.1 Create a directory using mkdir system call. 18
  - 2.2.2 Delete a directory using rmdir system call. 19
  - 2.2.3 Print the current working directory using getcwd system call. 19
  - 2.2.4 List directory contents using readdir system call. 20
- 2.3 KEYLOGGER OPERATIONS 21
  - 2.3.1 Open log file, write capture keystroke and closes the file descriptor after completing the logging operations using file I/O system call 22
  - 2.3.2 Using tcgetattr system call to modify the terminal behavior for capturing raw keystrokes and input system call getchar to captures a single character input from the user in real-time 22

3.0 TASK B: TESTING AND EVALUATION 23

- 3.1 FILE OPERATION 23
  - 3.1.1 Create or Open File 24
  - 3.1.2 Delete a File 24
  - 3.1.3 Read a File 24
  - 3.1.4 Write into a File 24
  - 3.1.5 Change Permissions of a File 24
- 3.2 DIRECTORY OPERATION 26
  - 3.2.1 Create a Directory command 26
  - 3.2.2 Delete a Directory command 26
  - 3.2.3 Print Current Working Directory command 26
  - 3.2.4 List Directory Content command.. 26

3.3 KEYLOGGER OPERATION 28
3.3.1 Keylogger Operation Command 28
3.4 MAN PAGE 29
4.0 TASK C: REFLECTION 31
4.1 USAGE OF CHATGPT AS A TOOL TO HELP STUDENTS OR PROGRAMMERS 31
4.2 TASK DISTRIBUTION 32
4.3 GROUP DISCUSSION COMMUNICATION MEDIUM 33
5.0 TASK D: USING GITHUB 34
6.0 CONCLUSION 35
APPENDIX 36

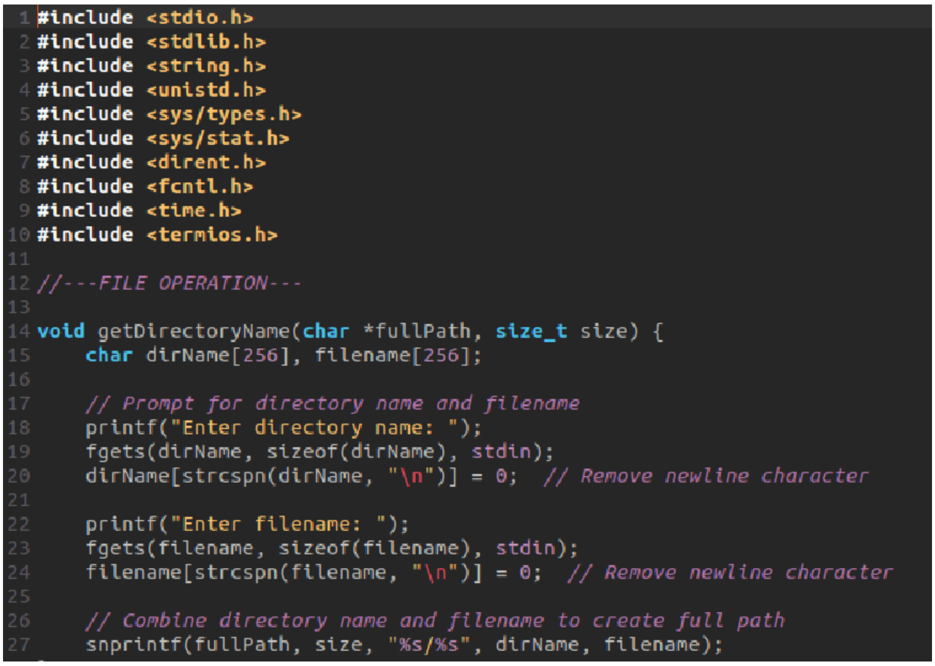
1.0 INTRODUCTION

The primary objective of this project is to build the Supercommand C program, a fully functional command-line tool that optimizes crucial system processes. The program includes a keylogger, directory management, and file operations as its primary functions. All of these functions have been set together for providing an integrated solution for system-level operations, including input by users accessible via both menu-based and command-line modes.

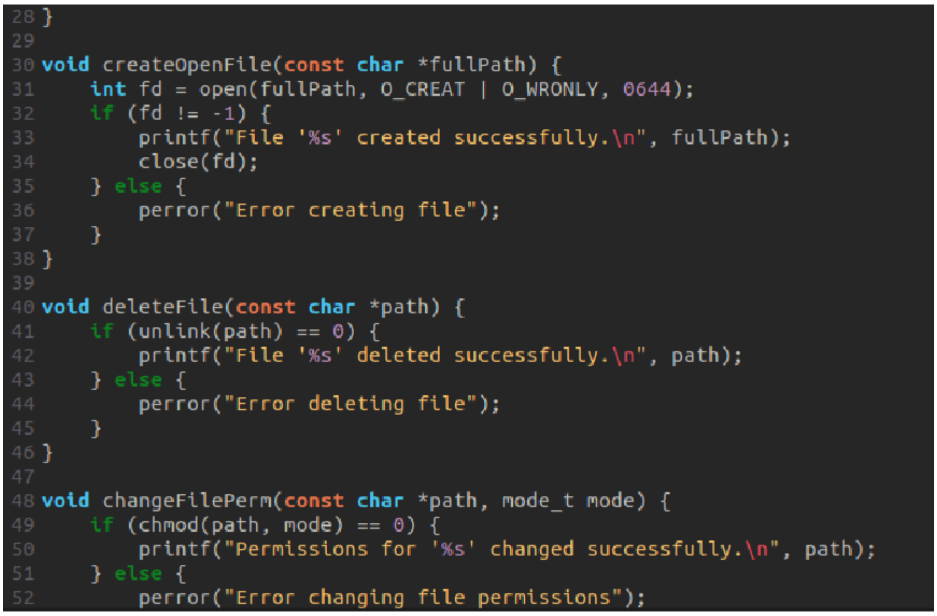
The file operations module facilitates it being convenient for users to add, open, read, write, remove, and change file permissions. The functionality is very beneficial for properly handling and modifying file-based resources. The directory operations module guarantees efficient file system arrangement and navigation by emphasizing adding and removing directories, displaying directory contents, and obtaining the currently operating directory. Finally, showcasing system monitoring performance, the keylogger module enables the ability of storing user keystrokes into a file, filled with timestamps, to record operations for testing or learning reasons.

The report is broken down into four essential parts. System calls and error handling are included in Task A, along with the comprehensive execution of the programming methods required for the previously mentioned processes. Task B details the testing and analysis of these functions, validating their accuracy and resiliency in many different circumstances. Task C assesses the cooperation, particularly how tasks were split up, how teammates communicated, and how ChatGPT assisted with coding tasks. Additionally, Task D indicates how GitHub is employed for keeping track of versions, allowing efficient teamwork and successful phases of development monitoring. Through this project, our group evaluated complex coding concepts, methods to solving issues, and collaborative strategies with the aim of developing a system management solution that functioned.

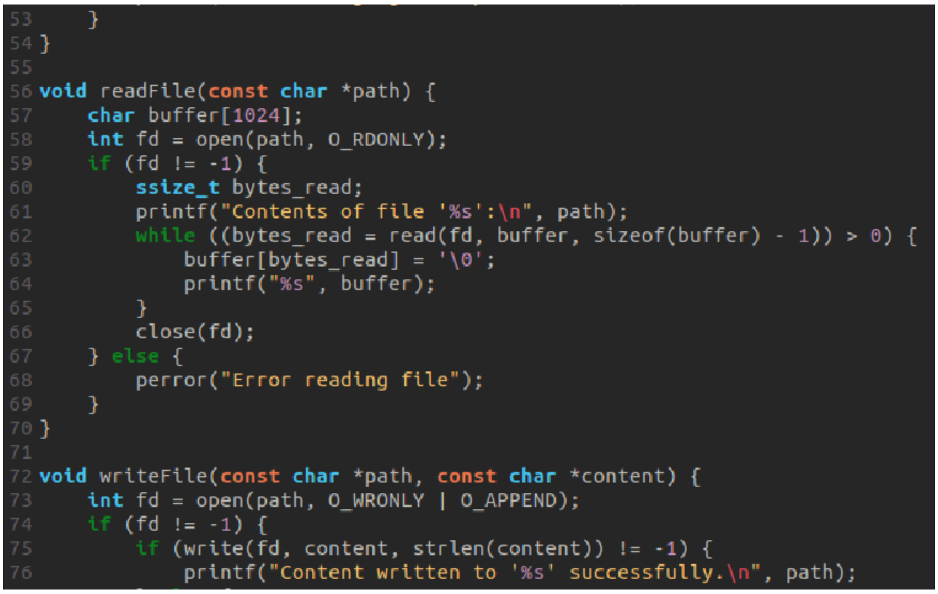
2.0 TASK A: CODING TASK



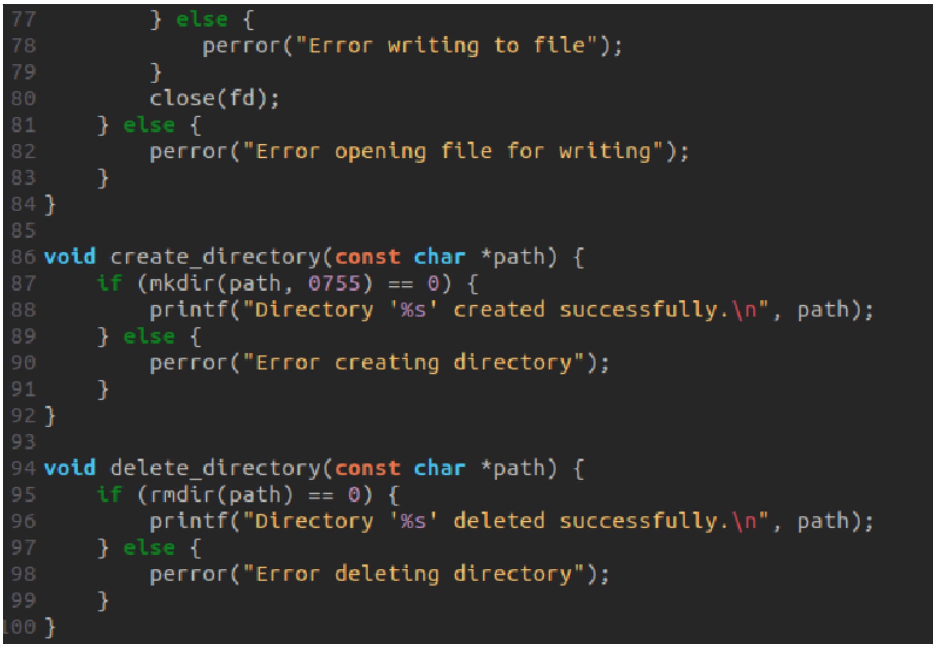
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <time.h>
#include <termios.h>

//---FILE OPERATION---

void getDirectoryName(char *fullPath, size_t size) {
    char dirName[256], filename[256];

    // Prompt for directory name and filename
    printf("Enter directory name: ");
    fgets(dirName, sizeof(dirName), stdin);
    dirName[strcspn(dirName, "\n")] = 0;  // Remove newline character

    printf("Enter filename: ");
    fgets(filename, sizeof(filename), stdin);
    filename[strcspn(filename, "\n")] = 0;  // Remove newline character

    // Combine directory name and filename to create full path
    snprintf(fullPath, size, "%s/%s", dirName, filename);
```



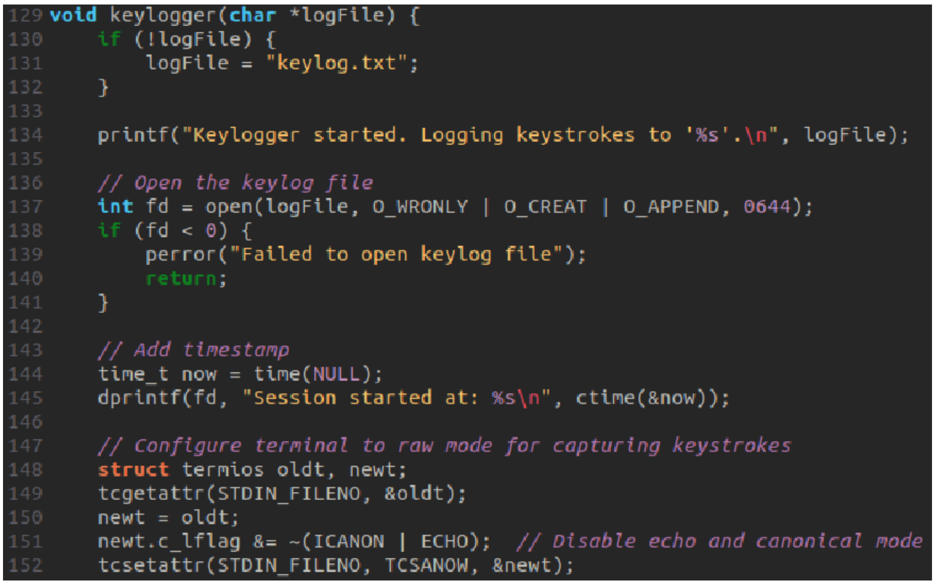
```c
}

void createOpenFile(const char *fullPath) {
    int fd = open(fullPath, O_CREAT | O_WRONLY, 0644);
    if (fd != -1) {
        printf("File '%s' created successfully.\n", fullPath);
        close(fd);
    } else {
        perror("Error creating file");
    }
}

void deleteFile(const char *path) {
    if (unlink(path) == 0) {
        printf("File '%s' deleted successfully.\n", path);
    } else {
        perror("Error deleting file");
    }
}

void changeFilePerm(const char *path, mode_t mode) {
    if (chmod(path, mode) == 0) {
        printf("Permissions for '%s' changed successfully.\n", path);
    } else {
        perror("Error changing file permissions");
```



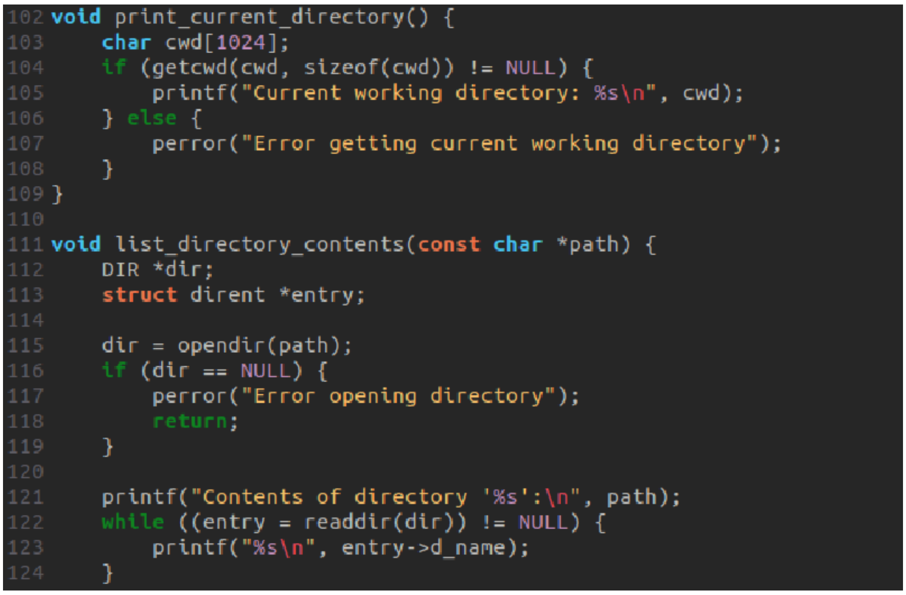
```
    }
}

void readFile(const char *path) {
    char buffer[1024];
    int fd = open(path, O_RDONLY);
    if (fd != -1) {
        ssize_t bytes_read;
        printf("Contents of file '%s':\n", path);
        while ((bytes_read = read(fd, buffer, sizeof(buffer) - 1)) > 0) {
            buffer[bytes_read] = '\0';
            printf("%s", buffer);
        }
        close(fd);
    } else {
        perror("Error reading file");
    }
}

void writeFile(const char *path, const char *content) {
    int fd = open(path, O_WRONLY | O_APPEND);
    if (fd != -1) {
        if (write(fd, content, strlen(content)) != -1) {
            printf("Content written to '%s' successfully.\n", path);
```



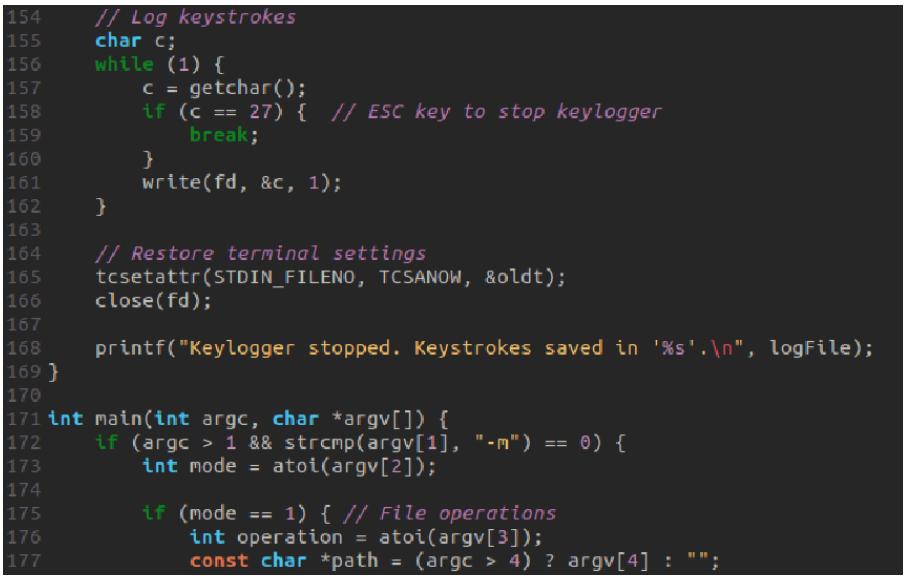
```
        } else {
            perror("Error writing to file");
        }
        close(fd);
    } else {
        perror("Error opening file for writing");
    }
}

void create_directory(const char *path) {
    if (mkdir(path, 0755) == 0) {
        printf("Directory '%s' created successfully.\n", path);
    } else {
        perror("Error creating directory");
    }
}

void delete_directory(const char *path) {
    if (rmdir(path) == 0) {
        printf("Directory '%s' deleted successfully.\n", path);
    } else {
        perror("Error deleting directory");
    }
}
```



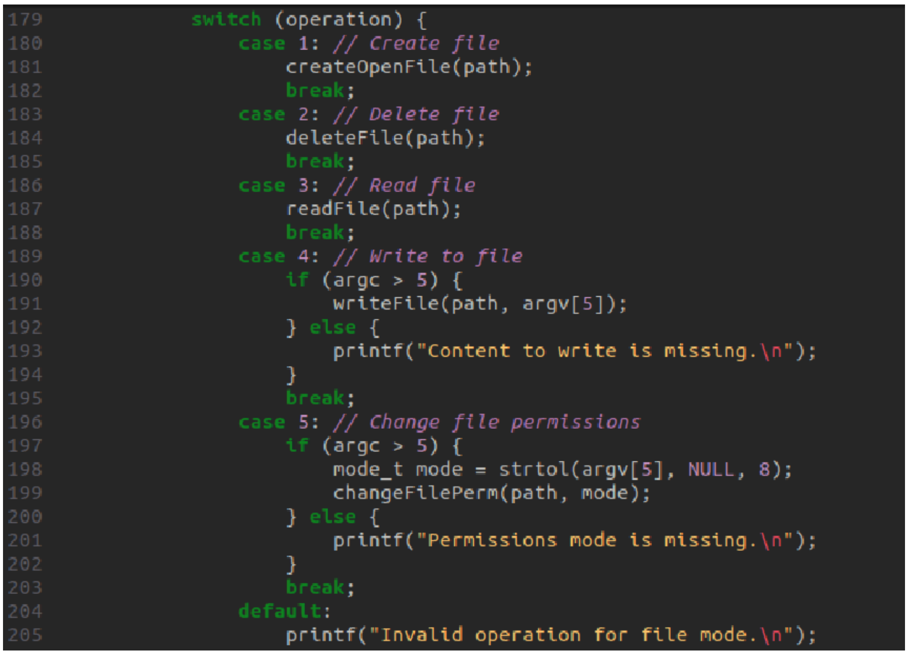
```
void keylogger(char *logFile) {
    if (!logFile) {
        logFile = "keylog.txt";
    }

    printf("Keylogger started. Logging keystrokes to '%s'.\n", logFile);

    // Open the keylog file
    int fd = open(logFile, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd < 0) {
        perror("Failed to open keylog file");
        return;
    }

    // Add timestamp
    time_t now = time(NULL);
    dprintf(fd, "Session started at: %s\n", ctime(&now));

    // Configure terminal to raw mode for capturing keystrokes
    struct termios oldt, newt;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);  // Disable echo and canonical mode
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
```



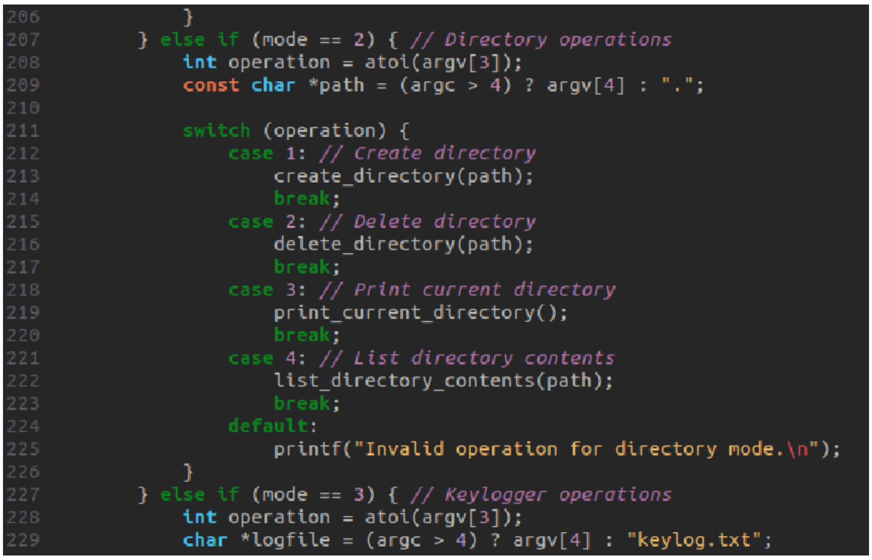
```c
void print_current_directory() {
    char cwd[1024];
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        printf("Current working directory: %s\n", cwd);
    } else {
        perror("Error getting current working directory");
    }
}

void list_directory_contents(const char *path) {
    DIR *dir;
    struct dirent *entry;

    dir = opendir(path);
    if (dir == NULL) {
        perror("Error opening directory");
        return;
    }

    printf("Contents of directory '%s':\n", path);
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }
```



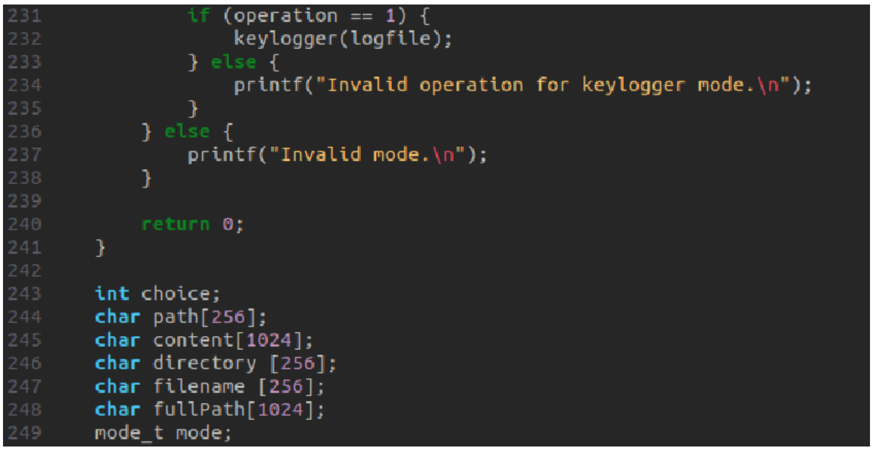
```c
    // Log keystrokes
    char c;
    while (1) {
        c = getchar();
        if (c == 27) {  // ESC key to stop keylogger
            break;
        }
        write(fd, &c, 1);
    }

    // Restore terminal settings
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    close(fd);

    printf("Keylogger stopped. Keystrokes saved in '%s'.\n", logFile);
}

int main(int argc, char *argv[]) {
    if (argc > 1 && strcmp(argv[1], "-m") == 0) {
        int mode = atoi(argv[2]);

        if (mode == 1) { // File operations
            int operation = atoi(argv[3]);
            const char *path = (argc > 4) ? argv[4] : "";
```



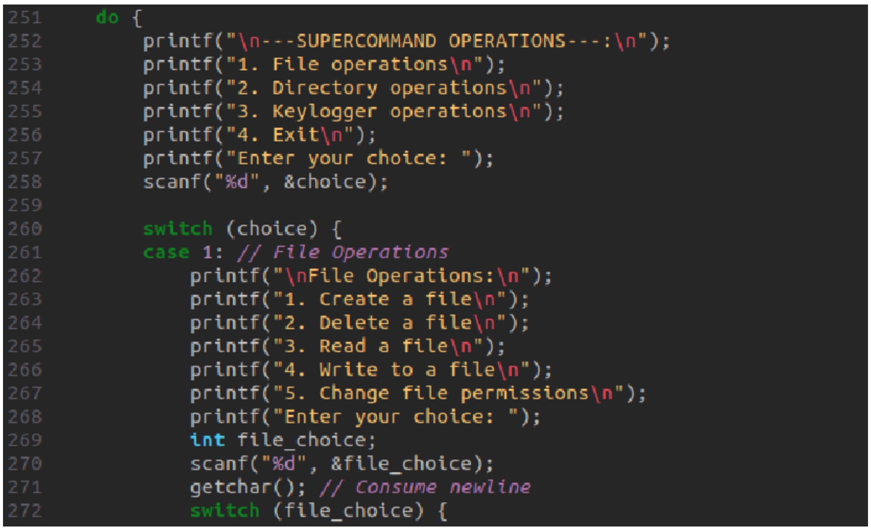
```c
            switch (operation) {
                case 1: // Create file
                    createOpenFile(path);
                    break;
                case 2: // Delete file
                    deleteFile(path);
                    break;
                case 3: // Read file
                    readFile(path);
                    break;
                case 4: // Write to file
                    if (argc > 5) {
                        writeFile(path, argv[5]);
                    } else {
                        printf("Content to write is missing.\n");
                    }
                    break;
                case 5: // Change file permissions
                    if (argc > 5) {
                        mode_t mode = strtol(argv[5], NULL, 8);
                        changeFilePerm(path, mode);
                    } else {
                        printf("Permissions mode is missing.\n");
                    }
                    break;
                default:
                    printf("Invalid operation for file mode.\n");
```



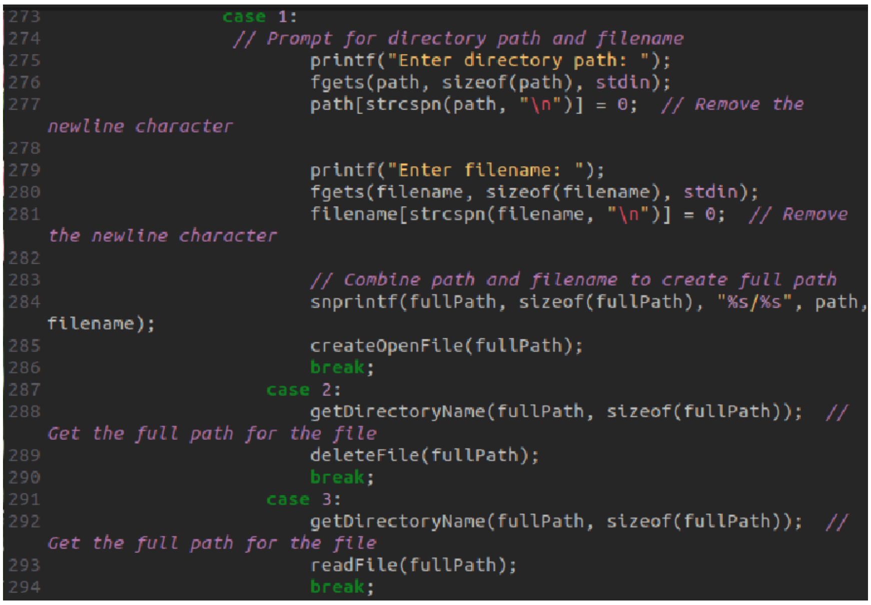
```
            }
        } else if (mode == 2) { // Directory operations
            int operation = atoi(argv[3]);
            const char *path = (argc > 4) ? argv[4] : ".";

            switch (operation) {
                case 1: // Create directory
                    create_directory(path);
                    break;
                case 2: // Delete directory
                    delete_directory(path);
                    break;
                case 3: // Print current directory
                    print_current_directory();
                    break;
                case 4: // List directory contents
                    list_directory_contents(path);
                    break;
                default:
                    printf("Invalid operation for directory mode.\n");
            }
        } else if (mode == 3) { // Keylogger operations
            int operation = atoi(argv[3]);
            char *logfile = (argc > 4) ? argv[4] : "keylog.txt";
```



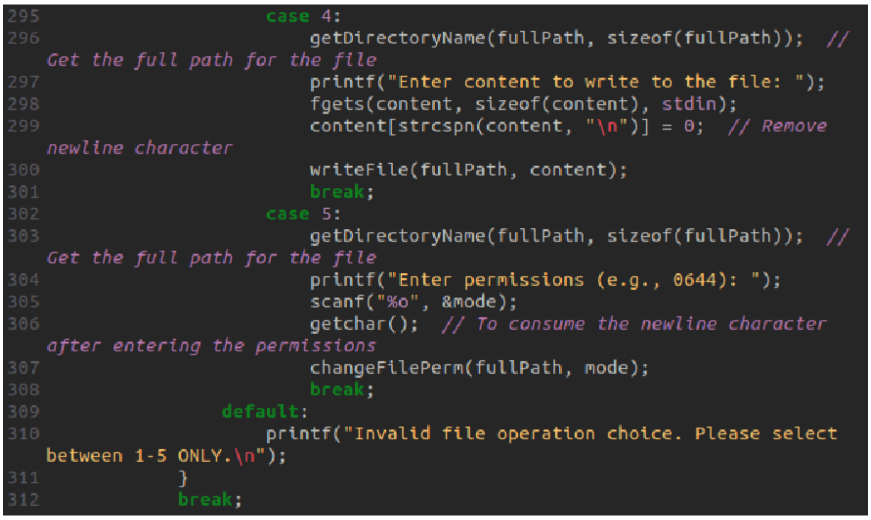
```
            if (operation == 1) {
                keylogger(logfile);
            } else {
                printf("Invalid operation for keylogger mode.\n");
            }
        } else {
            printf("Invalid mode.\n");
        }

        return 0;
    }

    int choice;
    char path[256];
    char content[1024];
    char directory [256];
    char filename [256];
    char fullPath[1024];
    mode_t mode;
```



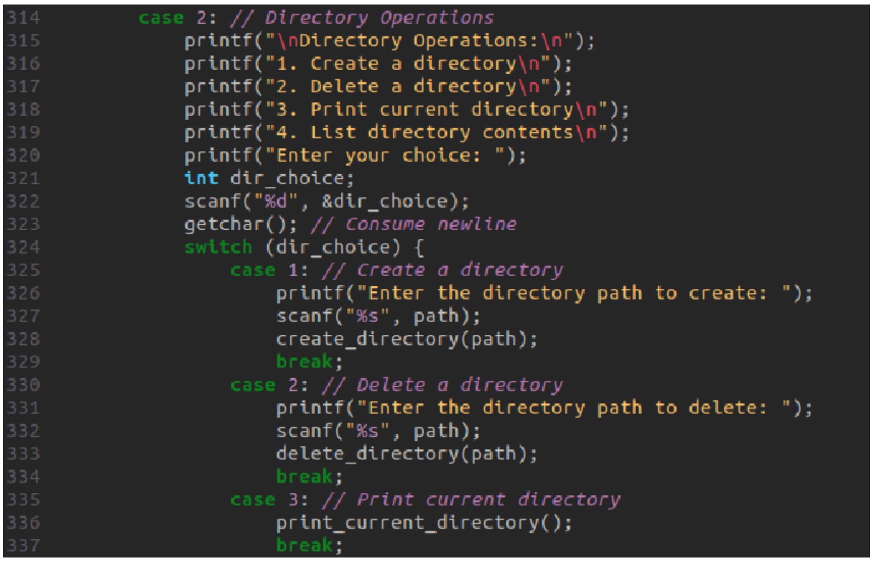
```
    do {
        printf("\n---SUPERCOMMAND OPERATIONS---:\n");
        printf("1. File operations\n");
        printf("2. Directory operations\n");
        printf("3. Keylogger operations\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1: // File Operations
            printf("\nFile Operations:\n");
            printf("1. Create a file\n");
            printf("2. Delete a file\n");
            printf("3. Read a file\n");
            printf("4. Write to a file\n");
            printf("5. Change file permissions\n");
            printf("Enter your choice: ");
            int file_choice;
            scanf("%d", &file_choice);
            getchar(); // Consume newline
            switch (file_choice) {
```



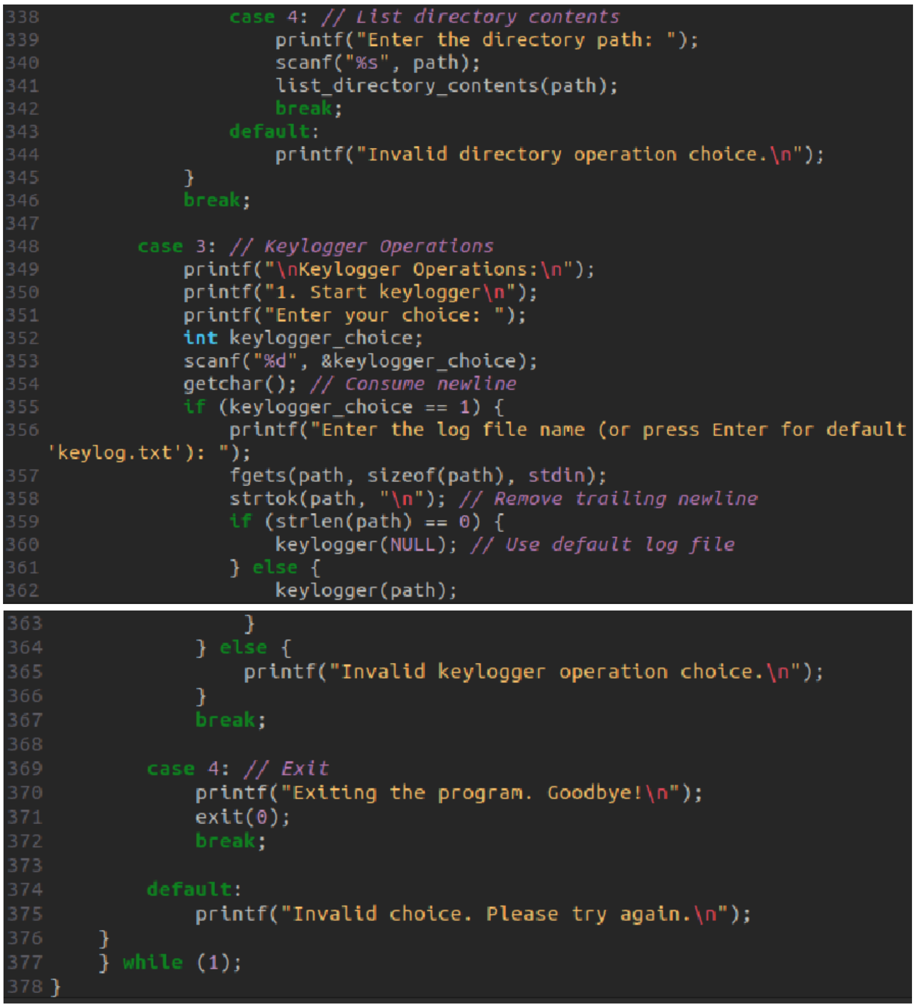
```
                case 1:
                 // Prompt for directory path and filename
                        printf("Enter directory path: ");
                        fgets(path, sizeof(path), stdin);
                        path[strcspn(path, "\n")] = 0;  // Remove the newline character

                        printf("Enter filename: ");
                        fgets(filename, sizeof(filename), stdin);
                        filename[strcspn(filename, "\n")] = 0;  // Remove the newline character

                        // Combine path and filename to create full path
                        snprintf(fullPath, sizeof(fullPath), "%s/%s", path, filename);
                        createOpenFile(fullPath);
                        break;
                    case 2:
                        getDirectoryName(fullPath, sizeof(fullPath));  // Get the full path for the file
                        deleteFile(fullPath);
                        break;
                    case 3:
                        getDirectoryName(fullPath, sizeof(fullPath));  // Get the full path for the file
                        readFile(fullPath);
                        break;
```



```
                    case 4:
                        getDirectoryName(fullPath, sizeof(fullPath));  // Get the full path for the file
                        printf("Enter content to write to the file: ");
                        fgets(content, sizeof(content), stdin);
                        content[strcspn(content, "\n")] = 0;  // Remove newline character
                        writeFile(fullPath, content);
                        break;
                    case 5:
                        getDirectoryName(fullPath, sizeof(fullPath));  // Get the full path for the file
                        printf("Enter permissions (e.g., 0644): ");
                        scanf("%o", &mode);
                        getchar();  // To consume the newline character after entering the permissions
                        changeFilePerm(fullPath, mode);
                        break;
                default:
                    printf("Invalid file operation choice. Please select between 1-5 ONLY.\n");
            }
            break;
```



```
        case 2: // Directory Operations
            printf("\nDirectory Operations:\n");
            printf("1. Create a directory\n");
            printf("2. Delete a directory\n");
            printf("3. Print current directory\n");
            printf("4. List directory contents\n");
            printf("Enter your choice: ");
            int dir_choice;
            scanf("%d", &dir_choice);
            getchar(); // Consume newline
            switch (dir_choice) {
                case 1: // Create a directory
                    printf("Enter the directory path to create: ");
                    scanf("%s", path);
                    create_directory(path);
                    break;
                case 2: // Delete a directory
                    printf("Enter the directory path to delete: ");
                    scanf("%s", path);
                    delete_directory(path);
                    break;
                case 3: // Print current directory
                    print_current_directory();
                    break;
```

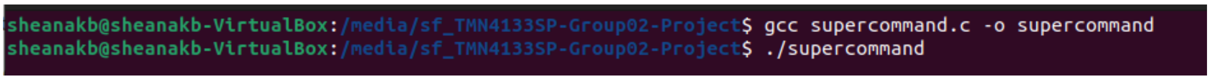


```
                case 4: // List directory contents
                    printf("Enter the directory path: ");
                    scanf("%s", path);
                    list_directory_contents(path);
                    break;
                default:
                    printf("Invalid directory operation choice.\n");
            }
            break;

        case 3: // Keylogger Operations
            printf("\nKeylogger Operations:\n");
            printf("1. Start keylogger\n");
            printf("Enter your choice: ");
            int keylogger_choice;
            scanf("%d", &keylogger_choice);
            getchar(); // Consume newline
            if (keylogger_choice == 1) {
                printf("Enter the log file name (or press Enter for default 'keylog.txt'): ");
                fgets(path, sizeof(path), stdin);
                strtok(path, "\n"); // Remove trailing newline
                if (strlen(path) == 0) {
                    keylogger(NULL); // Use default log file
                } else {
                    keylogger(path);
                }
            } else {
                printf("Invalid keylogger operation choice.\n");
            }
            break;

        case 4: // Exit
            printf("Exiting the program. Goodbye!\n");
            exit(0);
            break;

        default:
            printf("Invalid choice. Please try again.\n");
    }
    } while (1);
}
```

Figure 1: Full source code

2.1 COMPILING AND EXECUTING SUPERCOMMAND.C

The program supercommand.c was compiled using a gcc compiler in the Linux terminal. Once the program was compiled and returned zero errors, it was executed by running *./supercommand* in the Linux terminal. Figure 1 below shows the compilation and execution of our supercommand.c program.

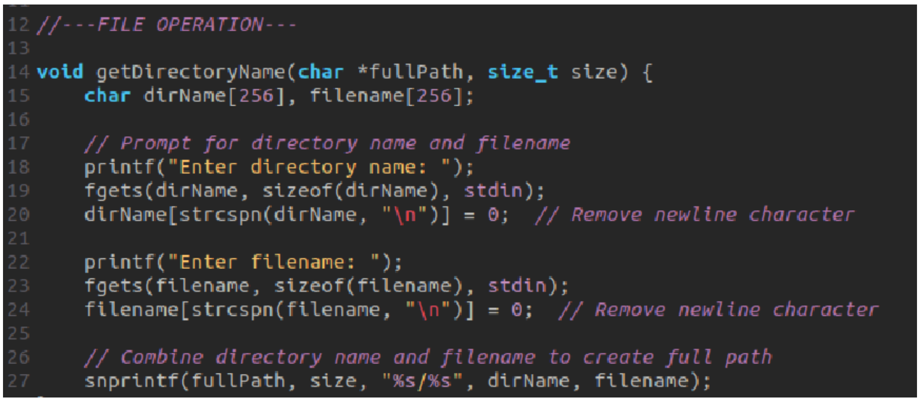


```
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ gcc supercommand.c -o supercommand
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand
```

Figure 2: Compiling and executing supercommand.c

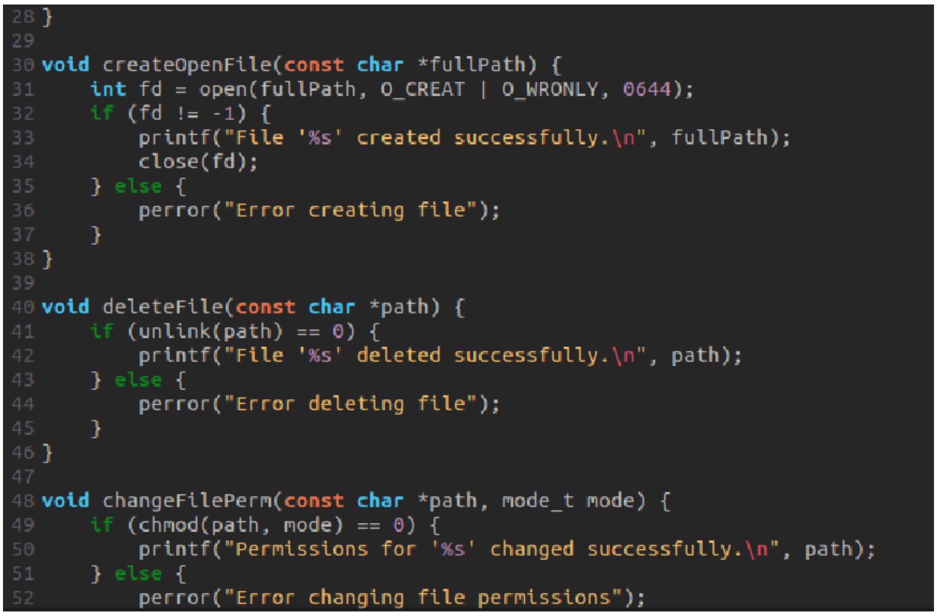
2.2 FILE OPERATIONS

Figures 3, 4 and 5 represents the functions required for the File Operation, meanwhile Figures 6, 7 and 8 shows the source code for File Operations via the menu-based system that prompts users to select the tasks to be executed.



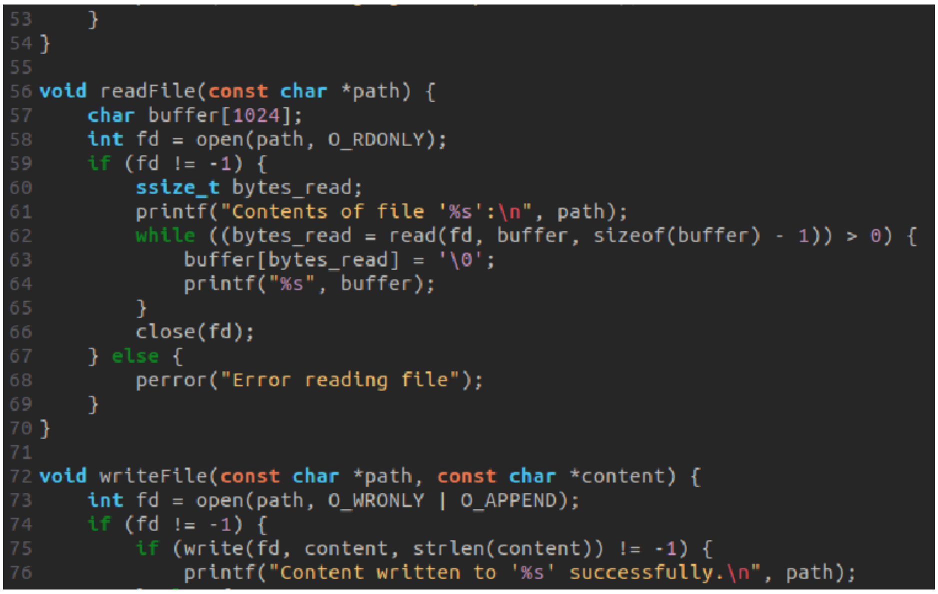
```c
//---FILE OPERATION---

void getDirectoryName(char *fullPath, size_t size) {
    char dirName[256], filename[256];

    // Prompt for directory name and filename
    printf("Enter directory name: ");
    fgets(dirName, sizeof(dirName), stdin);
    dirName[strcspn(dirName, "\n")] = 0;  // Remove newline character

    printf("Enter filename: ");
    fgets(filename, sizeof(filename), stdin);
    filename[strcspn(filename, "\n")] = 0;  // Remove newline character

    // Combine directory name and filename to create full path
    snprintf(fullPath, size, "%s/%s", dirName, filename);
```

Figure 3: Function getDirectoryName for File Operation



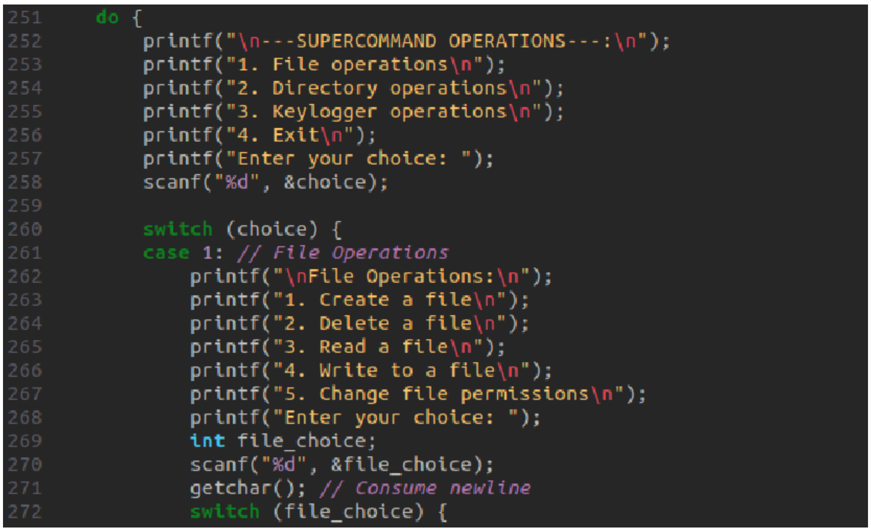
```c
}

void createOpenFile(const char *fullPath) {
    int fd = open(fullPath, O_CREAT | O_WRONLY, 0644);
    if (fd != -1) {
        printf("File '%s' created successfully.\n", fullPath);
        close(fd);
    } else {
        perror("Error creating file");
    }
}

void deleteFile(const char *path) {
    if (unlink(path) == 0) {
        printf("File '%s' deleted successfully.\n", path);
    } else {
        perror("Error deleting file");
    }
}

void changeFilePerm(const char *path, mode_t mode) {
    if (chmod(path, mode) == 0) {
        printf("Permissions for '%s' changed successfully.\n", path);
    } else {
        perror("Error changing file permissions");
```

Figure 4: Function createOpenFile, deleteFile, and changeFilePerm for File Operation



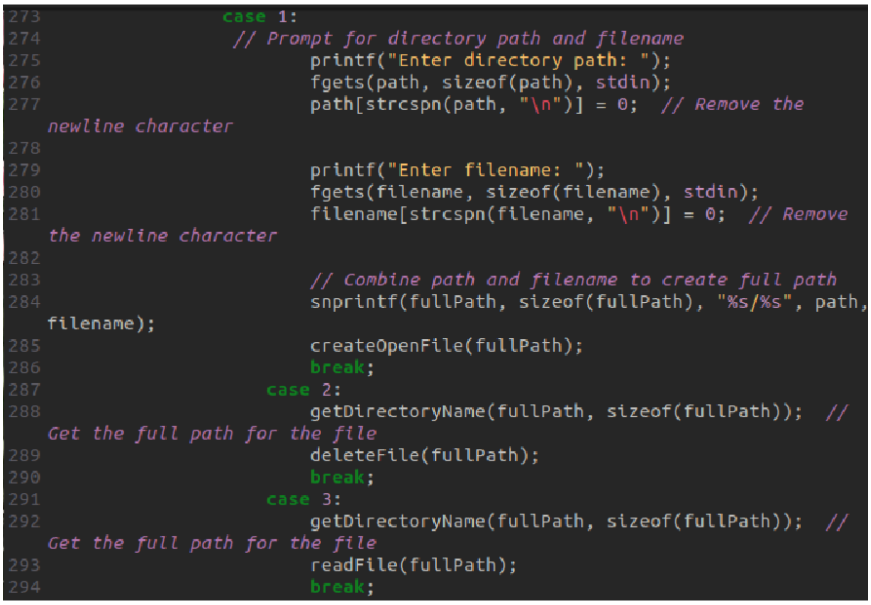
```c
    }
}

void readFile(const char *path) {
    char buffer[1024];
    int fd = open(path, O_RDONLY);
    if (fd != -1) {
        ssize_t bytes_read;
        printf("Contents of file '%s':\n", path);
        while ((bytes_read = read(fd, buffer, sizeof(buffer) - 1)) > 0) {
            buffer[bytes_read] = '\0';
            printf("%s", buffer);
        }
        close(fd);
    } else {
        perror("Error reading file");
    }
}

void writeFile(const char *path, const char *content) {
    int fd = open(path, O_WRONLY | O_APPEND);
    if (fd != -1) {
        if (write(fd, content, strlen(content)) != -1) {
            printf("Content written to '%s' successfully.\n", path);
```

Figure 5: Function readFile and writeFile for File Operation



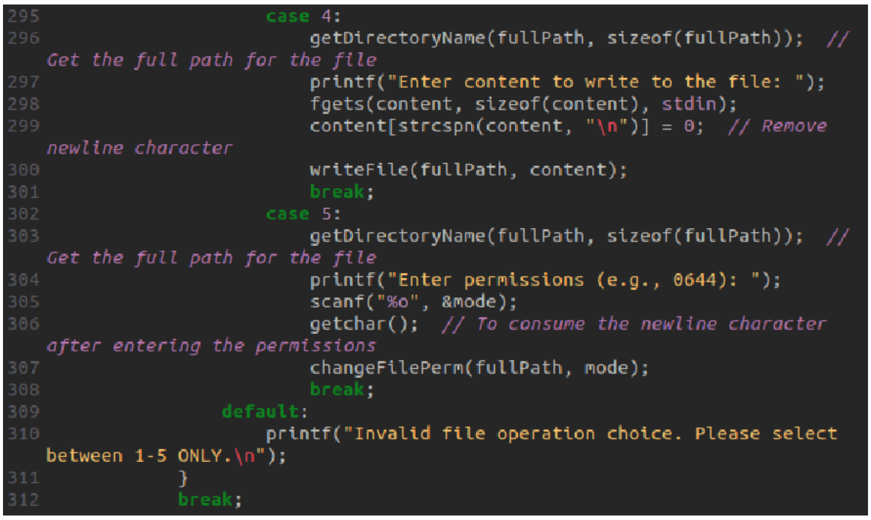
```c
    do {
        printf("\n---SUPERCOMMAND OPERATIONS---:\n");
        printf("1. File operations\n");
        printf("2. Directory operations\n");
        printf("3. Keylogger operations\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1: // File Operations
            printf("\nFile Operations:\n");
            printf("1. Create a file\n");
            printf("2. Delete a file\n");
            printf("3. Read a file\n");
            printf("4. Write to a file\n");
            printf("5. Change file permissions\n");
            printf("Enter your choice: ");
            int file_choice;
            scanf("%d", &file_choice);
            getchar(); // Consume newline
            switch (file_choice) {
```

Figure 6: Source code for menu-based system for File Operation execution



```
                case 1:
                 // Prompt for directory path and filename
                        printf("Enter directory path: ");
                        fgets(path, sizeof(path), stdin);
                        path[strcspn(path, "\n")] = 0;  // Remove the
newline character

                        printf("Enter filename: ");
                        fgets(filename, sizeof(filename), stdin);
                        filename[strcspn(filename, "\n")] = 0;  // Remove
the newline character

                        // Combine path and filename to create full path
                        snprintf(fullPath, sizeof(fullPath), "%s/%s", path,
filename);
                        createOpenFile(fullPath);
                        break;
                    case 2:
                        getDirectoryName(fullPath, sizeof(fullPath));  //
Get the full path for the file
                        deleteFile(fullPath);
                        break;
                    case 3:
                        getDirectoryName(fullPath, sizeof(fullPath));  //
Get the full path for the file
                        readFile(fullPath);
                        break;
```

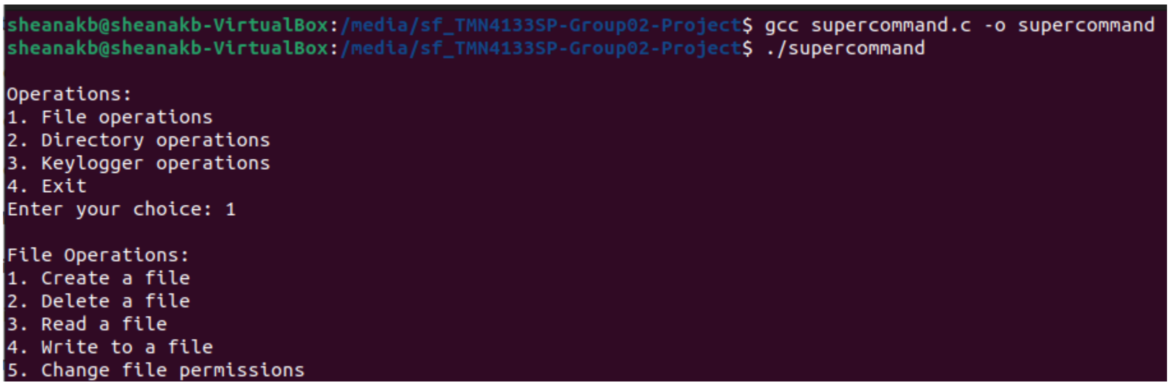
Figure 7: Switch case for File Operation selection



```
                    case 4:
                        getDirectoryName(fullPath, sizeof(fullPath));  //
Get the full path for the file
                        printf("Enter content to write to the file: ");
                        fgets(content, sizeof(content), stdin);
                        content[strcspn(content, "\n")] = 0;  // Remove
newline character
                        writeFile(fullPath, content);
                        break;
                    case 5:
                        getDirectoryName(fullPath, sizeof(fullPath));  //
Get the full path for the file
                        printf("Enter permissions (e.g., 0644): ");
                        scanf("%o", &mode);
                        getchar();  // To consume the newline character
after entering the permissions
                        changeFilePerm(fullPath, mode);
                        break;
                default:
                    printf("Invalid file operation choice. Please select
between 1-5 ONLY.\n");
            }
            break;
```

Figure 8: Continuation of switch case for File Operation selection

Figure 9 below shows the execution of supercommand.c from a menu-based perspective. Users will be prompted to choose which operation they want to execute. Upon selecting option 1, which is the File Operations option, users are required to select the next task they want to execute. These tasks consist of creating or opening a file, deleting a file, reading a file, writing to a file, and changing the file permissions.

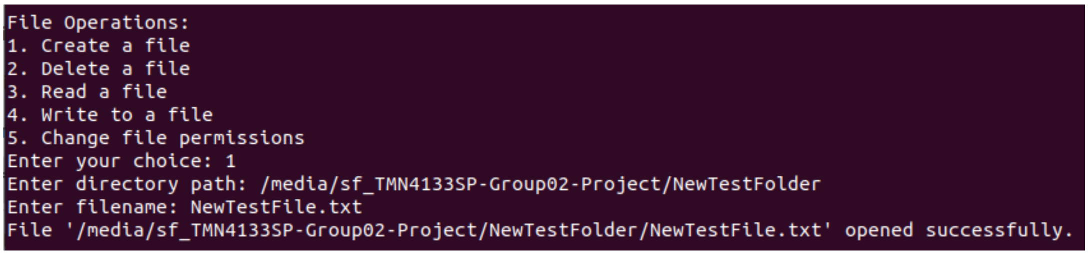


```
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ gcc supercommand.c -o supercommand
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand

Operations:
1. File operations
2. Directory operations
3. Keylogger operations
4. Exit
Enter your choice: 1

File Operations:
1. Create a file
2. Delete a file
3. Read a file
4. Write to a file
5. Change file permissions
```

Figure 9: Menu-based system execution for supercommand.c

2.2.1 Create/Open a File using open() system call

Selecting option 1 will lead users to create or open a file. Firstly, users will be required to enter the directory path for the file they want to create. Next, they are required to enter the file name to be created. If it opens successfully, a success statement which consists of the file name and path will be printed. This can be seen in Figure 10 below. Figure 11 shows the file that was successfully created in the desired directory.



```
File Operations:
1. Create a file
2. Delete a file
3. Read a file
4. Write to a file
5. Change file permissions
Enter your choice: 1
Enter directory path: /media/sf_TMN4133SP-Group02-Project/NewTestFolder
Enter filename: NewTestFile.txt
File '/media/sf_TMN4133SP-Group02-Project/NewTestFolder/NewTestFile.txt' opened successfully.
```

Figure 10: Creating a new file using the File Operation

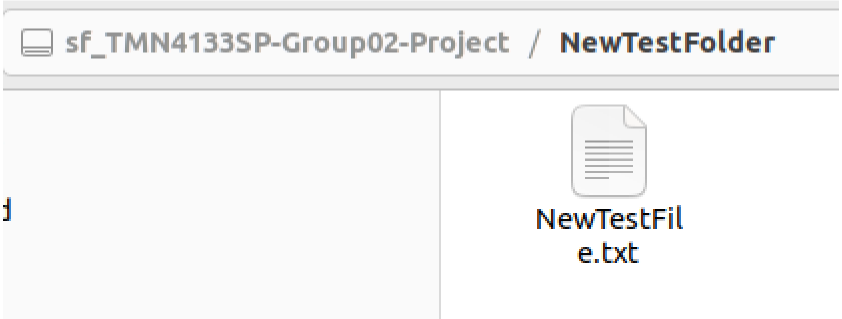
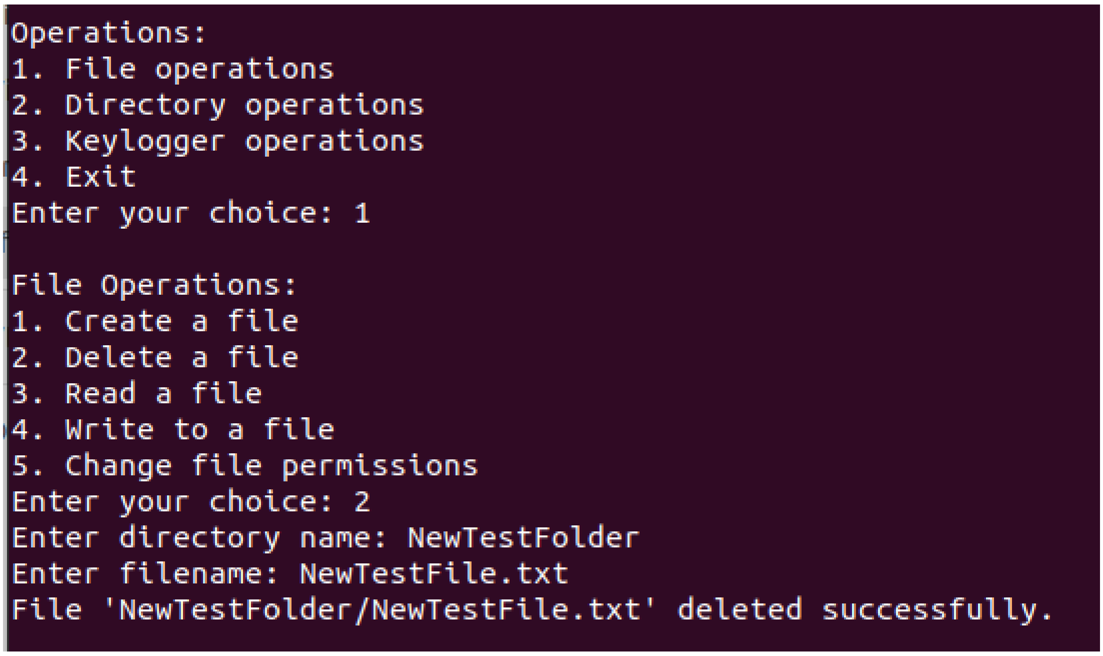


Figure 11: NewTestFile.txt successfully created in the desired directory

2.2.2 Deleting a File using unlink() system call

Figure 12 below shows the File Operation function to delete a file. Users will be prompted to enter the directory name and file name. A successful deletion will print out a deleted successfully statement containing the directory name and file name. Figure 13 shows the empty NewTestFolder folder after the NewTestFile.txt file is deleted.



```
Operations:
1. File operations
2. Directory operations
3. Keylogger operations
4. Exit
Enter your choice: 1

File Operations:
1. Create a file
2. Delete a file
3. Read a file
4. Write to a file
5. Change file permissions
Enter your choice: 2
Enter directory name: NewTestFolder
Enter filename: NewTestFile.txt
File 'NewTestFolder/NewTestFile.txt' deleted successfully.
```

Figure 12: Deleting a file from the menu-based system for File Operation

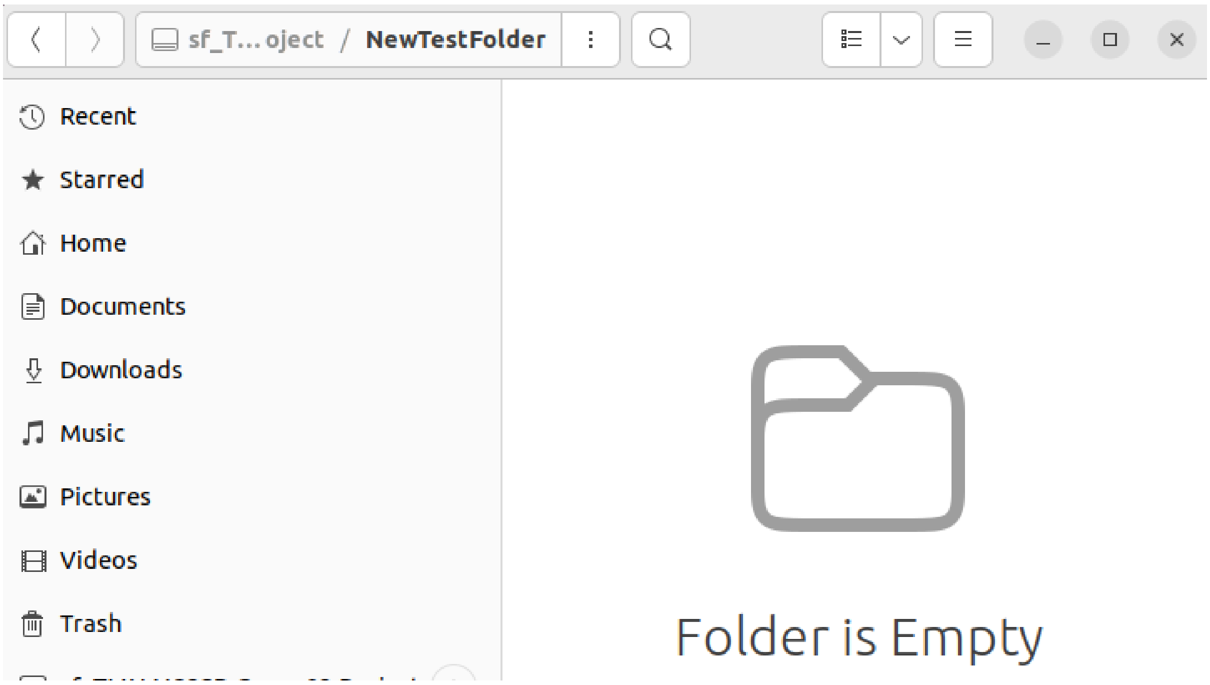
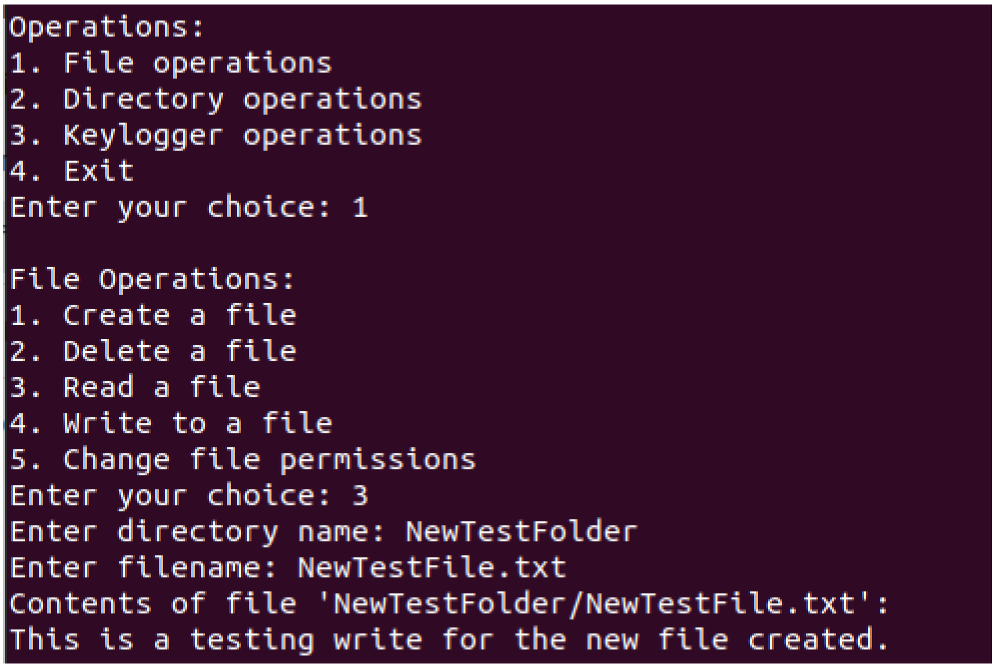


Figure 13: Empty NewTestFolder folder after deleting NewTestFile.txt

2.2.3 Reading a File using read() system call

When selecting option 3, which is to read a file, users are required to enter the directory name and file name of the desired file. Once entered, the contents of the file will be printed in the terminal as shown in Figure 14 below.

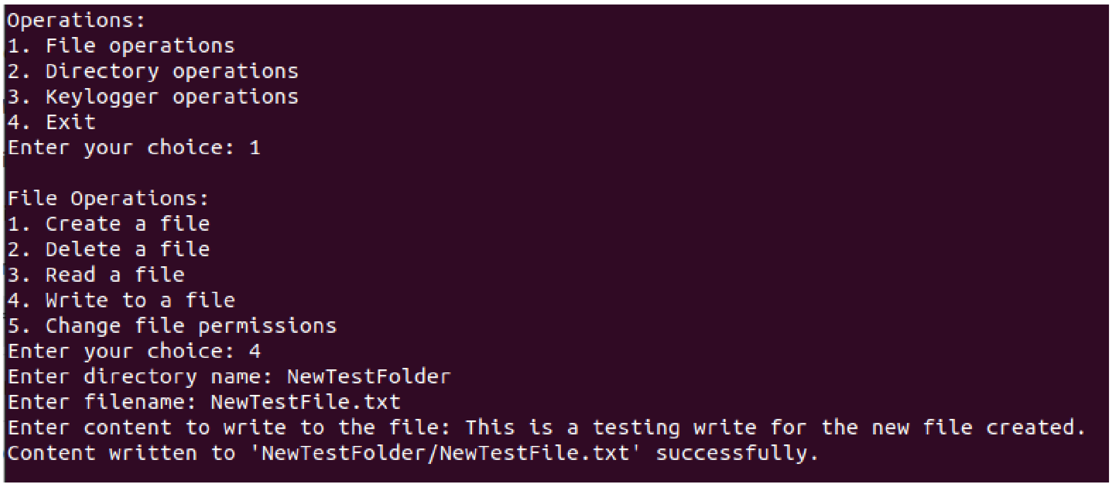


```
Operations:
1. File operations
2. Directory operations
3. Keylogger operations
4. Exit
Enter your choice: 1

File Operations:
1. Create a file
2. Delete a file
3. Read a file
4. Write to a file
5. Change file permissions
Enter your choice: 3
Enter directory name: NewTestFolder
Enter filename: NewTestFile.txt
Contents of file 'NewTestFolder/NewTestFile.txt':
This is a testing write for the new file created.
```

Figure 14: Reading the contents of NewTestFile.txt from menu-based system for File Operation

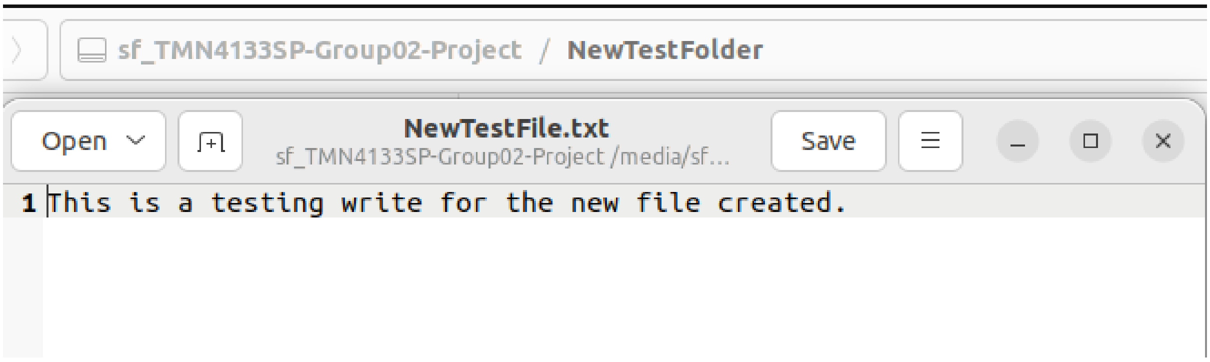
2.2.4 Writing into a File using write () system call

Users can write into their desired file by selecting option 4 from the File Operation menu. Once selected, users will be prompted to enter the directory and file name, to ensure the system captures the written contents in the correct file. Next, users can enter the content to be written into the desired file. A success statement will be printed if no error occurs as shown in Figure 15 below. Figure 16 shows the content being written into the desired file, this indicates the process was successfully written.



```
Operations:
1. File operations
2. Directory operations
3. Keylogger operations
4. Exit
Enter your choice: 1

File Operations:
1. Create a file
2. Delete a file
3. Read a file
4. Write to a file
5. Change file permissions
Enter your choice: 4
Enter directory name: NewTestFolder
Enter filename: NewTestFile.txt
Enter content to write to the file: This is a testing write for the new file created.
Content written to 'NewTestFolder/NewTestFile.txt' successfully.
```

Figure 15: Writing into file NewTestFile.txt from menu-based system for File Operation

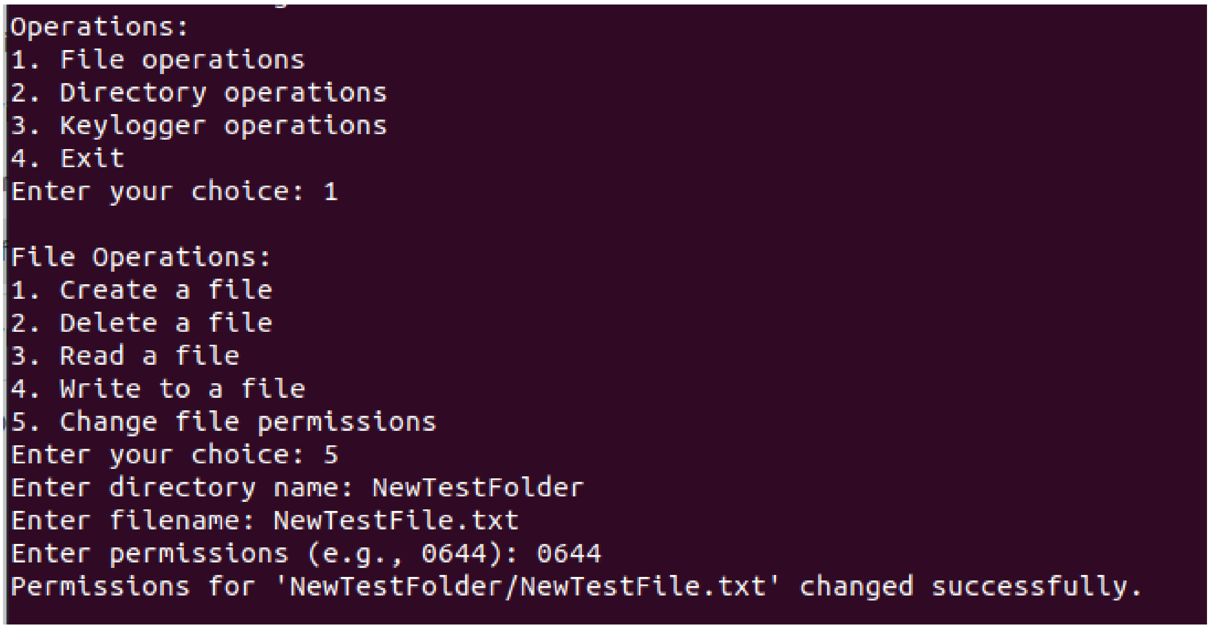

sf_TMN4133SP-Group02-Project / NewTestFolder
Open
NewTestFile.txt
sf_TMN4133SP-Group02-Project /media/sf...
Save

```
This is a testing write for the new file created.
```

Figure 16: Written content into NewTestFile.txt

2.2.5 Changing File Permissions using chmod() system call

When the user selects option 5 from the File Operation menu, they will be prompted to enter the directory and file name for the desired file they want to change permissions. The permissions can be changed using the numeric mode. In this example, we changed the file permission to read and write for user, and read only for both group and others. This is represented by the numeric mode which is 644. This can be seen in Figure 17 below.

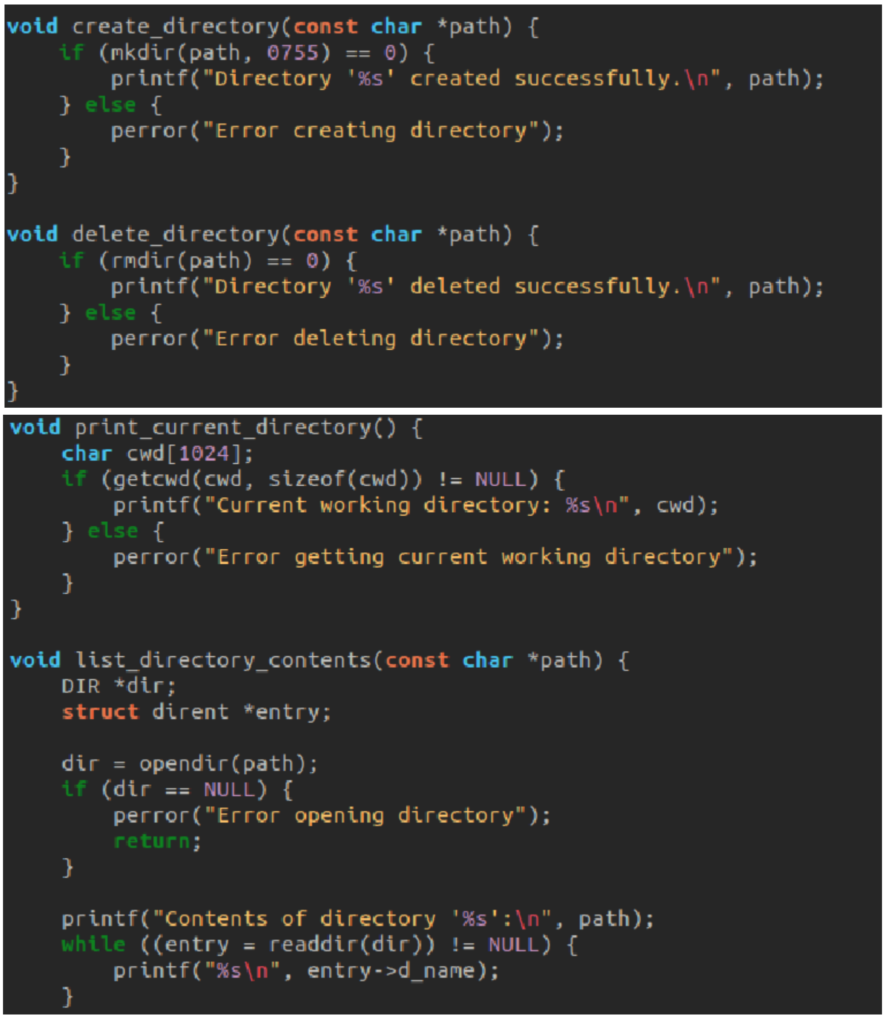


```
Operations:
1. File operations
2. Directory operations
3. Keylogger operations
4. Exit
Enter your choice: 1

File Operations:
1. Create a file
2. Delete a file
3. Read a file
4. Write to a file
5. Change file permissions
Enter your choice: 5
Enter directory name: NewTestFolder
Enter filename: NewTestFile.txt
Enter permissions (e.g., 0644): 0644
Permissions for 'NewTestFolder/NewTestFile.txt' changed successfully.
```

Figure 17: Changing file permission from menu-based system for File Operation

2.2 DIRECTORY OPERATIONS

The source code for integrating directory operations into the menu-based system is shown in Figure 18. It includes functions for creating directories, deleting directories, displaying the current working directory, and listing directory contents.



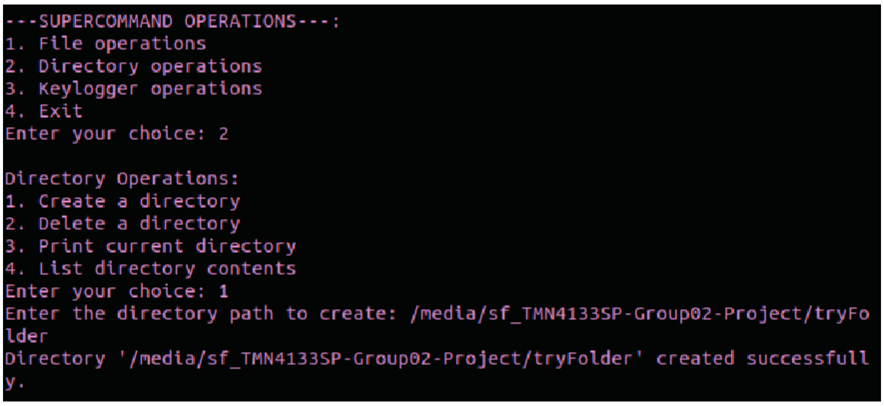
```
void create_directory(const char *path) {
    if (mkdir(path, 0755) == 0) {
        printf("Directory '%s' created successfully.\n", path);
    } else {
        perror("Error creating directory");
    }
}

void delete_directory(const char *path) {
    if (rmdir(path) == 0) {
        printf("Directory '%s' deleted successfully.\n", path);
    } else {
        perror("Error deleting directory");
    }
}
```

```
void print_current_directory() {
    char cwd[1024];
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        printf("Current working directory: %s\n", cwd);
    } else {
        perror("Error getting current working directory");
    }
}

void list_directory_contents(const char *path) {
    DIR *dir;
    struct dirent *entry;

    dir = opendir(path);
    if (dir == NULL) {
        perror("Error opening directory");
        return;
    }

    printf("Contents of directory '%s':\n", path);
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }
```

Figure 18: Source code for directory operations

2.2.1 Create a directory using mkdir system call.

To create a directory, the user needs to select option 2 for Directory Operations from the main menu. This will display five available choices for directory operations. To proceed, the user should select option 1 to create a directory. They will then be prompted to input the desired directory path. Once the path is provided, the directory will be created successfully as shown in Figure 19.

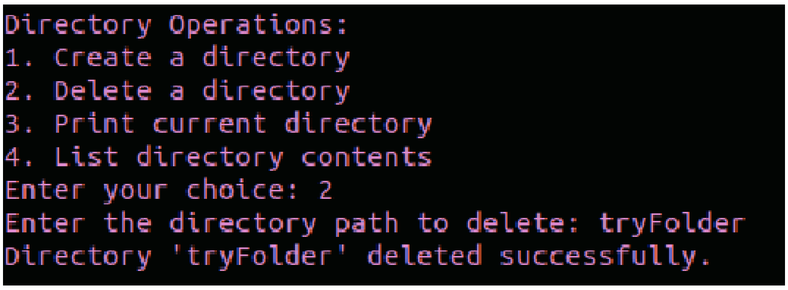


```
---SUPERCOMMAND OPERATIONS---:
1. File operations
2. Directory operations
3. Keylogger operations
4. Exit
Enter your choice: 2

Directory Operations:
1. Create a directory
2. Delete a directory
3. Print current directory
4. List directory contents
Enter your choice: 1
Enter the directory path to create: /media/sf_TMN4133SP-Group02-Project/tryFo
lder
Directory '/media/sf_TMN4133SP-Group02-Project/tryFolder' created successfull
y.
```

Figure 19: Output for create a directory

2.2.2 Delete a directory using rmdir system call.

To delete a directory, the user needs to select option 2 from the Directory Operations menu. They will then be prompted to input the path of the directory to be deleted. Once the correct path is provided, the directory will be successfully deleted.

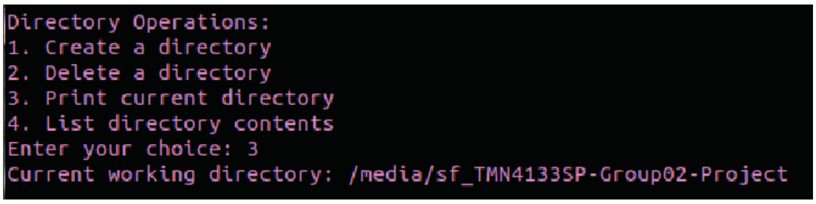


```
Directory Operations:
1. Create a directory
2. Delete a directory
3. Print current directory
4. List directory contents
Enter your choice: 2
Enter the directory path to delete: tryFolder
Directory 'tryFolder' deleted successfully.
```

Figure 20: Output for delete a directory

2.2.3 Print the current working directory using getcwd system call.

To print the current working directory, the user needs to select option 3 from the Directory Operations menu. After choosing this option, the program will display the absolute path of the directory where the program is currently being executed.

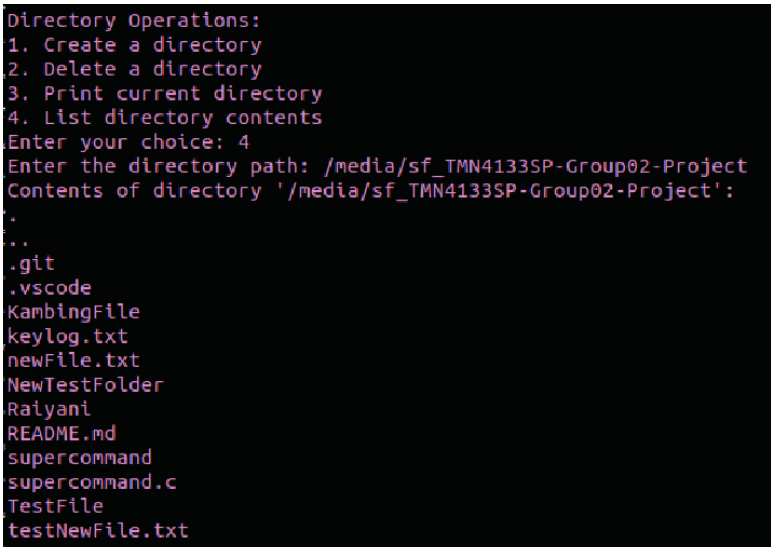


```
Directory Operations:
1. Create a directory
2. Delete a directory
3. Print current directory
4. List directory contents
Enter your choice: 3
Current working directory: /media/sf_TMN4133SP-Group02-Project
```

Figure 21: Output for print the current working directory

2.2.4 List directory contents using readdir system call.

To list the contents of a directory, the user needs to select option 4 from the Directory Operations menu. The user will then be prompted to input the path of the directory contents that they want to view. After providing the correct path, the program will display all files and folders inside the specified directory.

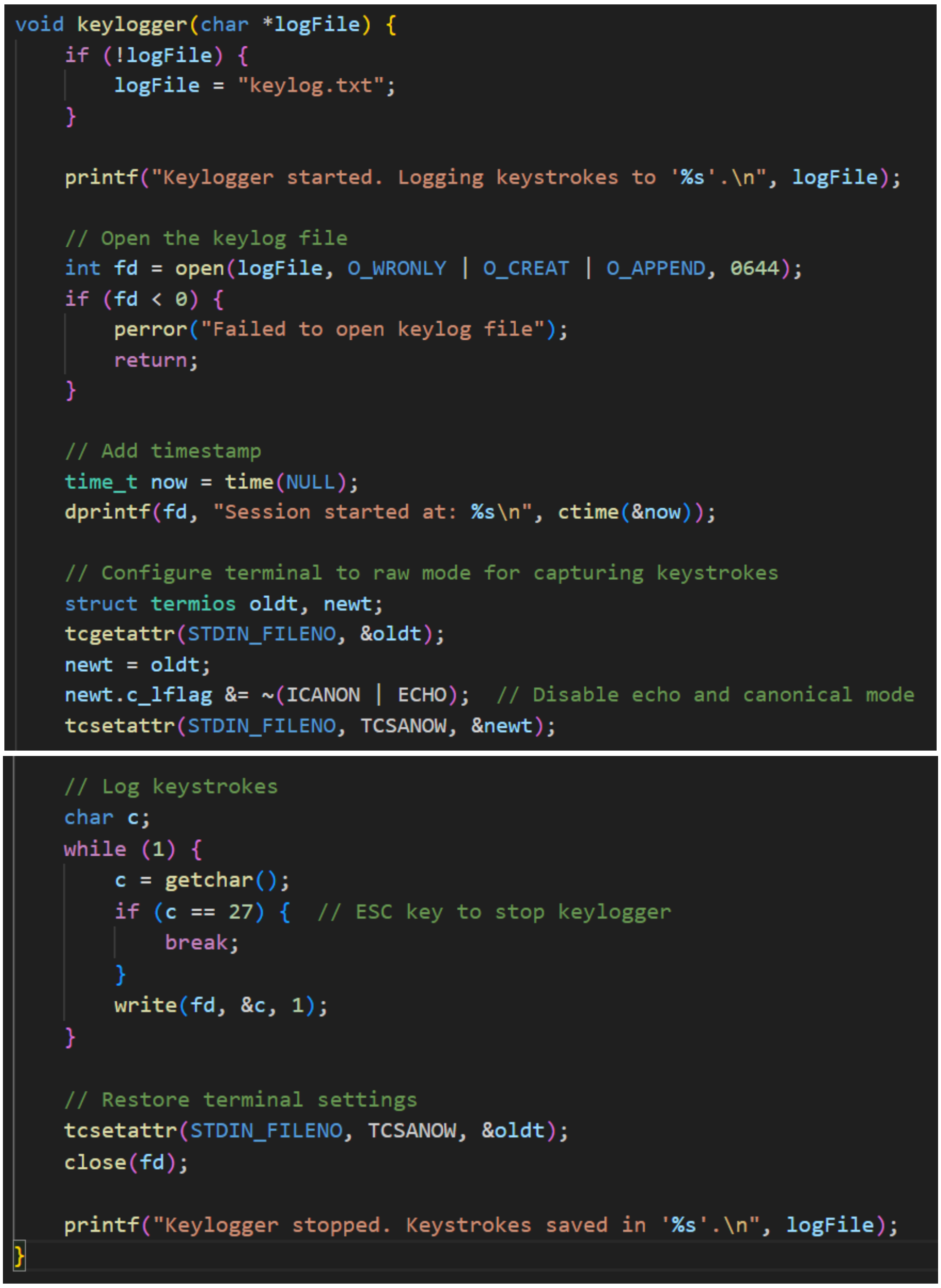


```
Directory Operations:
1. Create a directory
2. Delete a directory
3. Print current directory
4. List directory contents
Enter your choice: 4
Enter the directory path: /media/sf_TMN4133SP-Group02-Project
Contents of directory '/media/sf_TMN4133SP-Group02-Project':
.
..
.git
.vscode
KambingFile
keylog.txt
newFile.txt
NewTestFolder
Raiyani
README.md
supercommand
supercommand.c
TestFile
testNewFile.txt
```

Figure 22: Output for list directory contents

2.3 KEYLOGGER OPERATIONS

Figure 23 shows the source code for keylogger operations via the menu-based system.

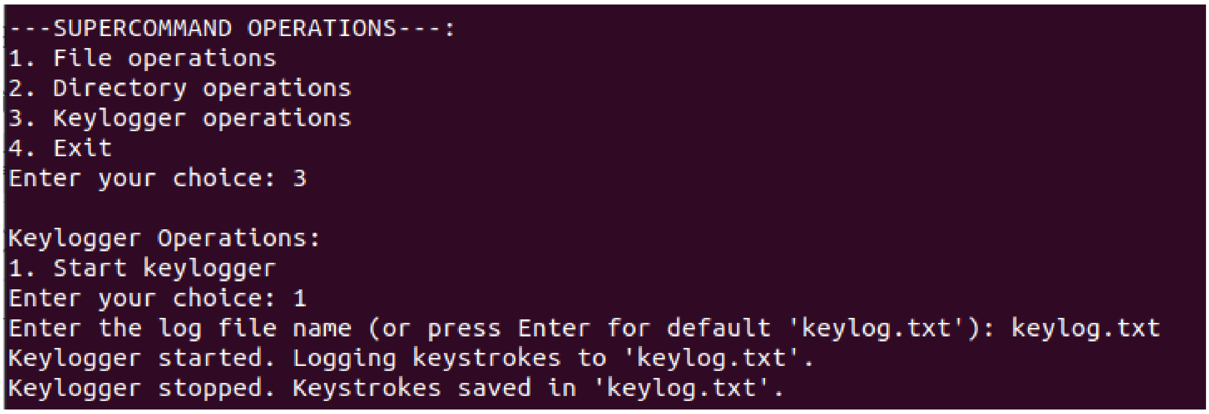


```
void keylogger(char *logFile) {
    if (!logFile) {
        logFile = "keylog.txt";
    }

    printf("Keylogger started. Logging keystrokes to '%s'.\n", logFile);

    // Open the keylog file
    int fd = open(logFile, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd < 0) {
        perror("Failed to open keylog file");
        return;
    }

    // Add timestamp
    time_t now = time(NULL);
    dprintf(fd, "Session started at: %s\n", ctime(&now));

    // Configure terminal to raw mode for capturing keystrokes
    struct termios oldt, newt;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);  // Disable echo and canonical mode
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);

    // Log keystrokes
    char c;
    while (1) {
        c = getchar();
        if (c == 27) {  // ESC key to stop keylogger
            break;
        }
        write(fd, &c, 1);
    }

    // Restore terminal settings
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    close(fd);

    printf("Keylogger stopped. Keystrokes saved in '%s'.\n", logFile);
}
```

Figure 23: Source code for Keylogger Operation

2.3.1 Open log file, write capture keystroke and closes the file descriptor after completing the logging operations using file I/O system call

As shown in Figure 24, choose option 3 to enable keylogger operations. Then choose option 1 to start keylogger operations and enter the log file name. Prompt any input, for example, "Hello World!" and press 'Esc' key once done.

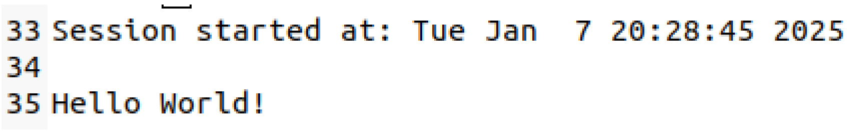


```
---SUPERCOMMAND OPERATIONS---:
1. File operations
2. Directory operations
3. Keylogger operations
4. Exit
Enter your choice: 3

Keylogger Operations:
1. Start keylogger
Enter your choice: 1
Enter the log file name (or press Enter for default 'keylog.txt'): keylog.txt
Keylogger started. Logging keystrokes to 'keylog.txt'.
Keylogger stopped. Keystrokes saved in 'keylog.txt'.
```

Figure 24: Menu-based for Keylogger Operation

2.3.2 Using tcgetattr system call to modify the terminal behavior for capturing raw keystrokes and input system call getchar to captures a single character input from the user in real-time

To view the previous entries and their timestamps, open the keylog.txt file as shown in Figure 25.



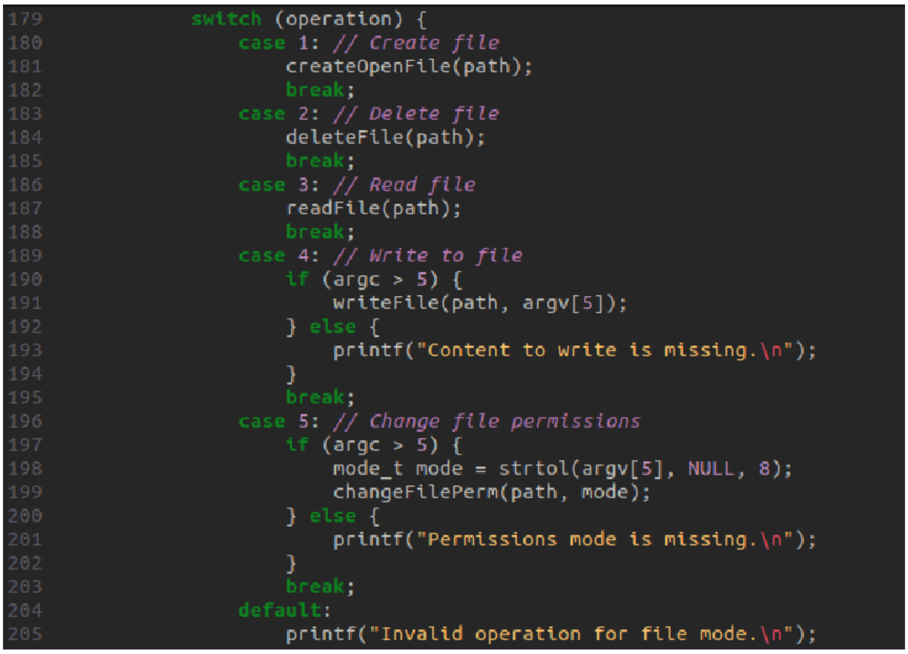
```
Session started at: Tue Jan  7 20:28:45 2025

Hello World!
```

Figure 25: keylog.txt content

3.0 TASK B: TESTING AND EVALUATION

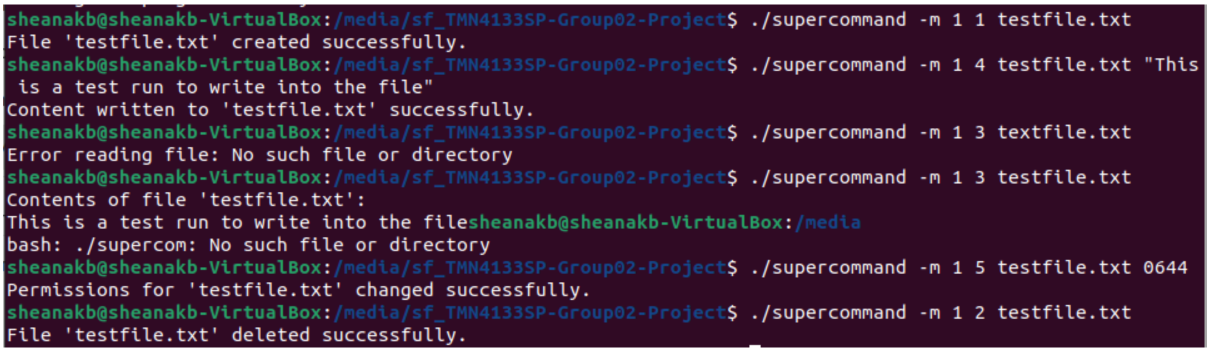
3.1 FILE OPERATION

Figure 26 shows the source code for the command-line based system for the File Operation. They contain the same functions as the menu-based system, which includes creating or opening a file, deleting a file, reading a file, writing into a file, and changing the file permissions. Each file operation is executed through specific command-line arguments, which allows the user to run them from the terminal directly.



```c
            switch (operation) {
                case 1: // Create file
                    createOpenFile(path);
                    break;
                case 2: // Delete file
                    deleteFile(path);
                    break;
                case 3: // Read file
                    readFile(path);
                    break;
                case 4: // Write to file
                    if (argc > 5) {
                        writeFile(path, argv[5]);
                    } else {
                        printf("Content to write is missing.\n");
                    }
                    break;
                case 5: // Change file permissions
                    if (argc > 5) {
                        mode_t mode = strtol(argv[5], NULL, 8);
                        changeFilePerm(path, mode);
                    } else {
                        printf("Permissions mode is missing.\n");
                    }
                    break;
                default:
                    printf("Invalid operation for file mode.\n");
```

Figure 26: Source code for command line-based system for File Operation



```
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 1 1 testfile.txt
File 'testfile.txt' created successfully.
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 1 4 testfile.txt "This
 is a test run to write into the file"
Content written to 'testfile.txt' successfully.
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 1 3 textfile.txt
Error reading file: No such file or directory
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 1 3 testfile.txt
Contents of file 'testfile.txt':
This is a test run to write into the filesheanakb@sheanakb-VirtualBox:/media
bash: ./supercom: No such file or directory
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 1 5 testfile.txt 0644
Permissions for 'testfile.txt' changed successfully.
sheanakb@sheanakb-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 1 2 testfile.txt
File 'testfile.txt' deleted successfully.
```

Figure 27: Command line-based system commands for File Operation

Figure 27 above shows the command line arguments that were executed for File Operations via the Linux terminal. The explanation for each parameter is as follows:

3.1.1 Create or Open File

```
./supercommand -m 1 1 filename.txt
```

This command is for creating or opening a file. The parameter *-m 1* represents mode 1, which is the File Operation, followed by operation number 1 for creating or opening a file, and lastly the name of the file to be created.

3.1.2 Delete a File

```
./supercommand -m 1 2 filename.txt
```

This command is used to delete a file. The parameter -m 1 represents mode 1, which is the File Operation, followed by operation number 2 for deleting a file. Then, the name of the file to be deleted.

3.1.3 Read a File

```
./supercommand -m 1 3 filename.txt
```

This command is used to read the contents of a file. The parameter -m 1 represents mode 1, which is the File Operation, followed by operation number 3 for reading a file, and lastly, the name of the file to be read.

3.1.4 Write into a File

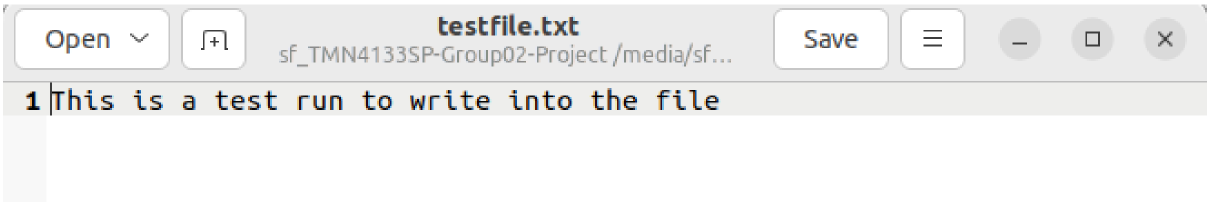
```
./supercommand -m 1 4 filename.txt “Content to write”
```

This command is used to write content into a file. The parameter -m 1 represents mode 1, which is the File Operation, followed by operation number 4 for writing into a file, and finally, the name of the file where the content will be written.

3.1.5 Change Permissions of a File

```
./supercommand -m 1 5 filename.txt 0644
```

This command is used to change the permissions of a file. The parameter -m 1 represents mode 1, which is the File Operation, followed by operation number 5 for modifying file permissions, and lastly, the name of the file whose permissions need to be updated.



```
This is a test run to write into the file
```

Figure 28: Written content into file testfile.txt from command line File Operation system

Figure 28 above shows the written content after running the command *./supercommand -m 1 4 testfile.txt "This is a test run to write into the file"*. The command successfully writes the content into the file name entered by the user. Meanwhile, Figure 29 below shows the contents of the directory after running *./supercommand -m 1 2 testfile.txt* where the file testfile.txt has been deleted.

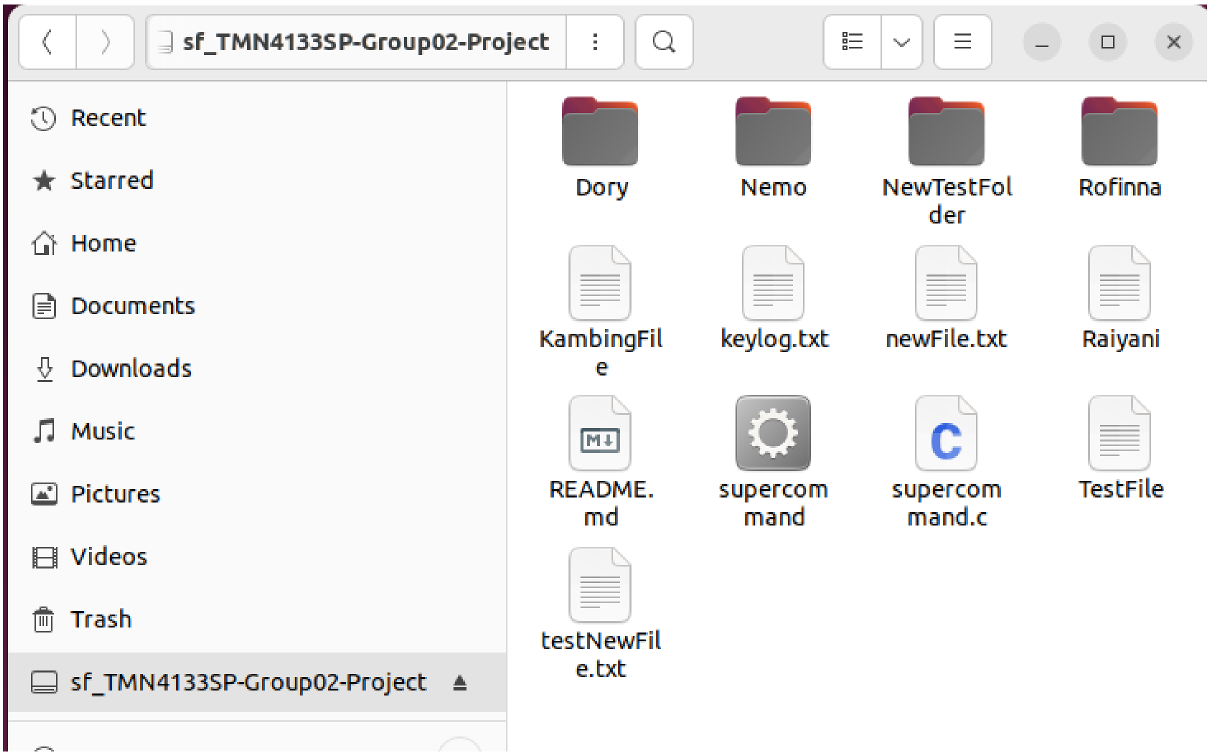
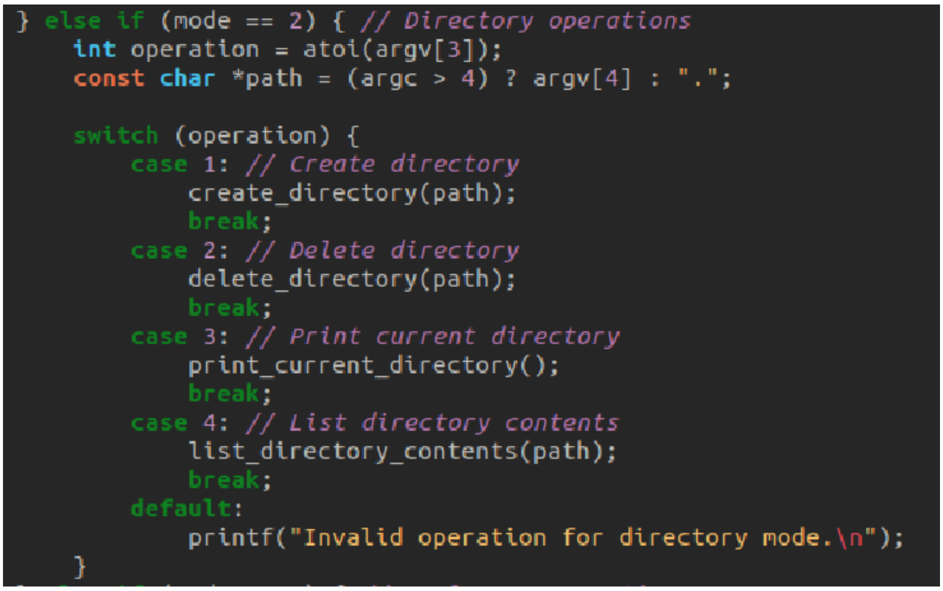


Figure 29: File testfile.txt deleted from the directory

3.2 DIRECTORY OPERATION

Figure 30 shows the source code for implementing directory operations in the command-line-based system. The code includes functions to create directories, delete directories, display the current working directory, and list the contents of a directory. Each operation is triggered by specific command-line arguments, allowing users to perform tasks directly from the terminal.



```
} else if (mode == 2) { // Directory operations
    int operation = atoi(argv[3]);
    const char *path = (argc > 4) ? argv[4] : ".";

    switch (operation) {
        case 1: // Create directory
            create_directory(path);
            break;
        case 2: // Delete directory
            delete_directory(path);
            break;
        case 3: // Print current directory
            print_current_directory();
            break;
        case 4: // List directory contents
            list_directory_contents(path);
            break;
        default:
            printf("Invalid operation for directory mode.\n");
    }
```

Figure 30: Source code for directory operations

3.2.1 Create a Directory command

```
./supercommand -m 2 1 ./new_folder
```

This command specifies mode -m 2 for directory operations, followed by operation 1 to create a directory. The path ./new_folder indicates where the new directory will be created.

3.2.2 Delete a Directory command

```
./supercommand -m 2 2 ./new_folder
```

In this command, mode -m 2 is used for directory operations, and operation 2 is specified to delete the directory. The path ./new_folder indicates the directory to be removed.

3.2.3 Print Current Working Directory command

```
./supercommand -m 2 3
```

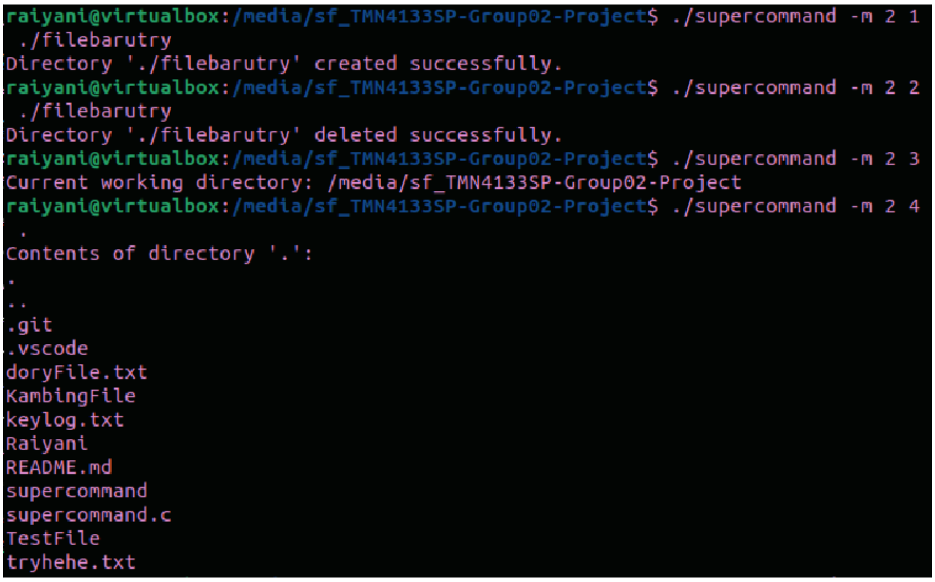
In this command, mode -m 2 is for directory operations, and operation 3 prints the current working directory. No additional path is required.

3.2.4 List Directory Content command

```
./supercommand -m 2 4 .
```

This command uses mode -m 2 for directory operations and operation 4 to list contents. The . indicates the current directory, but this can be replaced with any valid directory path.

Figure 31 displays the output for the command-line arguments used in the directory operations, including tasks such as creating, deleting, printing the current working directory, and listing directory contents.



```
raiyani@virtualbox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 2 1
 ./filebarutry
Directory './filebarutry' created successfully.
raiyani@virtualbox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 2 2
 ./filebarutry
Directory './filebarutry' deleted successfully.
raiyani@virtualbox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 2 3
Current working directory: /media/sf_TMN4133SP-Group02-Project
raiyani@virtualbox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 2 4
 .
Contents of directory '.':
.
..
.git
.vscode
doryFile.txt
KambingFile
keylog.txt
Raiyani
README.md
supercommand
supercommand.c
TestFile
tryhehe.txt
```

Figure 31. Output of command-line arguments for directory operations

3.3 KEYLOGGER OPERATION

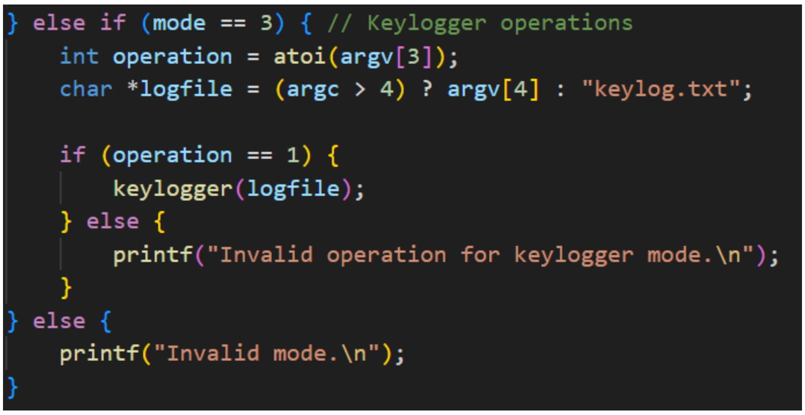
The purpose of this test is to evaluate the functionality of the keylogger operation in logging user keystrokes to a specified file.

3.3.1 Keylogger Operation Command

```
./supercommand -m 3 1 keylog.txt
```

In this command, mode -m 3 is for directory operations, operation 1 is associated with starting the keylogger and keylog.txt the name of the file where the captured keystrokes will be logged.

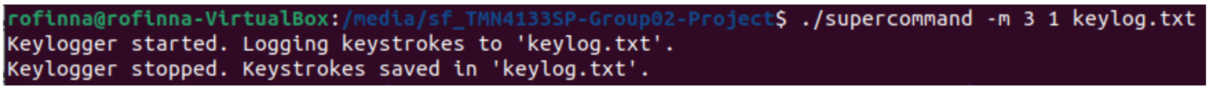
Figure 32 shows the source code of keylogger operations for command-line based system.



```
} else if (mode == 3) { // Keylogger operations
    int operation = atoi(argv[3]);
    char *logfile = (argc > 4) ? argv[4] : "keylog.txt";

    if (operation == 1) {
        keylogger(logfile);
    } else {
        printf("Invalid operation for keylogger mode.\n");
    }
} else {
    printf("Invalid mode.\n");
}
```

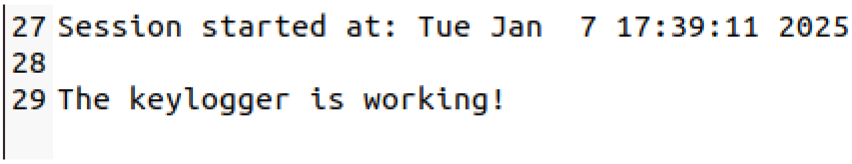
Figure 32: Source code for Command-line based system for keylogger operations

Figure 33 show the keylogger operation that are executed from command line. Command used: ./supercommand -m 3 1 keylog.txt and Figure 34 shows the output of keylogger operations in keylog.txt.



```
rofinna@rofinna-VirtualBox:/media/sf_TMN4133SP-Group02-Project$ ./supercommand -m 3 1 keylog.txt
Keylogger started. Logging keystrokes to 'keylog.txt'.
Keylogger stopped. Keystrokes saved in 'keylog.txt'.
```

Figure 33: Command-line Arguments for keylogger operations

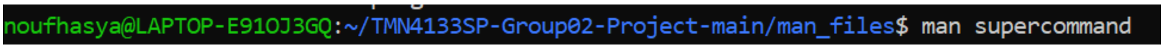


```
Session started at: Tue Jan  7 17:39:11 2025

The keylogger is working!
```

Figure 34: Output of keylogger operations in keylog.txt

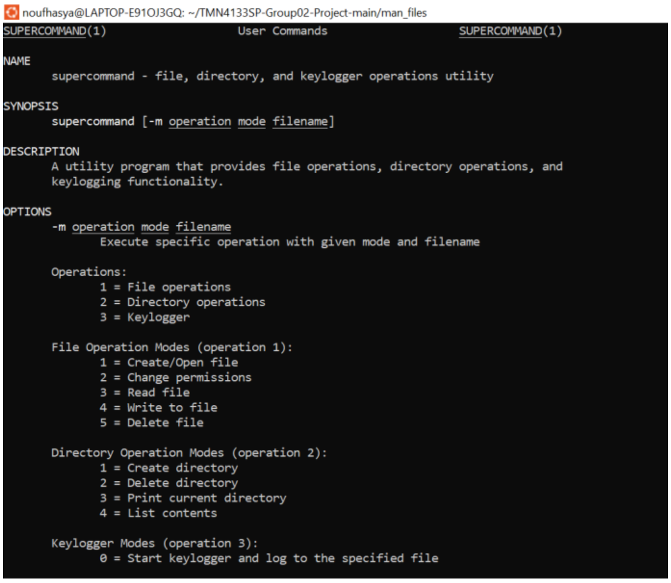
3.4 MAN PAGE

The supercommand utility is a comprehensive program designed to handle file operations, directory operations, and a keylogger functionality. The tool allows users to perform these operations through a menu-based system or by using command-line arguments for advanced use.



```
noufhasya@LAPTOP-E91OJ3GQ:~/TMN4133SP-Group02-Project-main/man_files$ man supercommand
```

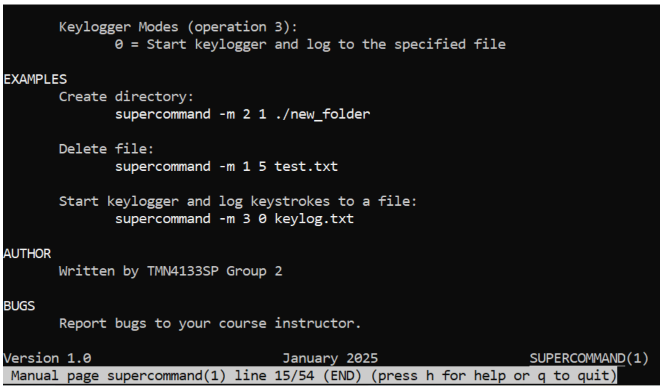
Figure 35: *man supercommand* is used to display man page



noufhasya@LAPTOP-E91OJ3GQ: ~/TMN4133SP-Group02-Project-main/man_files

```
SUPERCOMMAND(1)                  User Commands                 SUPERCOMMAND(1)

NAME
       supercommand - file, directory, and keylogger operations utility

SYNOPSIS
       supercommand [-m operation mode filename]

DESCRIPTION
       A utility program that provides file operations, directory operations, and
       keylogging functionality.

OPTIONS
       -m operation mode filename
              Execute specific operation with given mode and filename

       Operations:
              1 = File operations
              2 = Directory operations
              3 = Keylogger

       File Operation Modes (operation 1):
              1 = Create/Open file
              2 = Change permissions
              3 = Read file
              4 = Write to file
              5 = Delete file

       Directory Operation Modes (operation 2):
              1 = Create directory
              2 = Delete directory
              3 = Print current directory
              4 = List contents

       Keylogger Modes (operation 3):
              0 = Start keylogger and log to the specified file
```

Figure 36: The output of *man supercommand*, showing the overview section for the command.



```
       Keylogger Modes (operation 3):
              0 = Start keylogger and log to the specified file

EXAMPLES
       Create directory:
              supercommand -m 2 1 ./new_folder

       Delete file:
              supercommand -m 1 5 test.txt

       Start keylogger and log keystrokes to a file:
              supercommand -m 3 0 keylog.txt

AUTHOR
       Written by TMN4133SP Group 2

BUGS
       Report bugs to your course instructor.

Version 1.0                    January 2025                  SUPERCOMMAND(1)
 Manual page supercommand(1) line 15/54 (END) (press h for help or q to quit)
```

Figure 37: Continuation of the output of *man supercommand*, showing the overview section for the command.

The program supports three primary operations:

A. File Operations (Operation 1)

These operations allow users to manage files:

- **Mode 1**: Create or open a file.
- **Mode 2**: Change file permissions.
- **Mode 3**: Read and display file content.
- **Mode 4**: Write user input to a file.
- **Mode 5**: Delete a file.

B. Directory Operations (Operation 2)

These operations are designed for managing directories:

- **Mode 1**: Create a directory.
- **Mode 2**: Delete a directory.
- **Mode 3**: Print the current working directory.
- **Mode 4**: List the contents of a directory.

C. Keylogger (Operation 3)

This operation runs a keylogger in the background and logs keystrokes to a specified file. A timestamp is added at the start of each logging session.

4.0 TASK C: REFLECTION

4.1 USAGE OF CHATGPT AS A TOOL TO HELP STUDENTS OR PROGRAMMERS

ChatGPT has been an innovative technology that connects the learning process of students with the productivity of expert programmers. Assisting with programming problems has never been easier than with its rapid responses and fully functional code generator. When students use it, it's like having a personal tutor by their side, helping them out when other resources fall short.

The collaborative nature of ChatGPT is among its most beneficial features. Users are able to test out concepts, adjust, and solve problems in real time while using the tool for projects or learning new concepts. A better grasp of the material can be achieved by its ability to propose efficient methods, explain complex ideas, and even modify code to meet individual needs. Making programming more approachable is another way ChatGPT empowers users. For instance, it breaks down large projects into more manageable chunks, making things like creating command-line utilities or implementing system calls much easier. In addition to assisting users in developing self-assurance, this speeds up project completion. Additionally, it's a helpful tool for exploring fresh ideas, so users can find solutions they might not have thought of otherwise.

However, ChatGPT does have some restrictions that it must adhere to. Due to its lack of human intuition, ChatGPT may generate code that necessitates additional attention to ensure it meets real-world requirements. The fact that this is the case highlights how important it is to use ChatGPT as a guide rather than a solution that is definitive. Users are required to validate and test its outputs while simultaneously continuously improving their own skills.

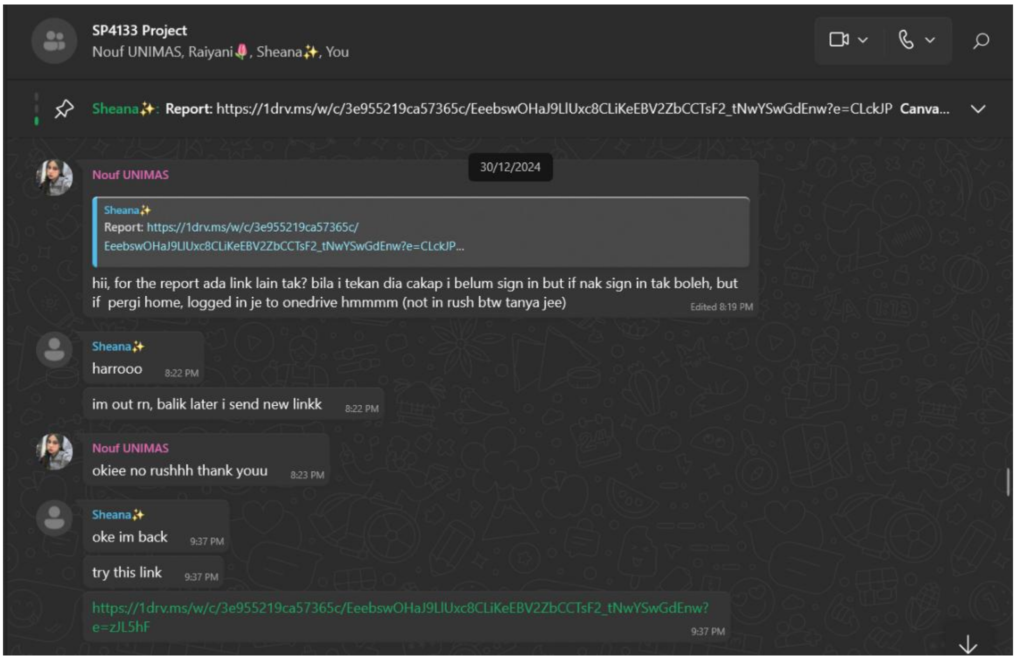
In conclusion, ChatGPT is an excellent tool for learning and getting tasks completed. In addition to fostering creativity and encouraging inquiry, it also assists users in approaching programming with a greater sense of comfort. Despite the fact that it necessitates the use of critical thinking in order to fully exploit its potential, it unquestionably makes the experience of coding more intriguing and exciting.

4.2 TASK DISTRIBUTION

Table 1. Task Distribution Table

Group Member	Tasks
Sheana Kasih Benedict (78525)	• Code for file operations • Task A o Create/Open a file o Change file permissions o Read a file and print contents o Write user input to a file o Delete a file • Task B o Implement command-line arguments for all file operations.
Nur Raiyani Binti Mohd Yusri Azhar (80685)	• Code for directory operations • Task A o Create a directory o Delete a directory o Print the current working directory o List directory contents • Task B o Implement command-line arguments for all directory operations.
Rofinna Ellya Embang Anak Umar @ Richard (81056)	• Code for keylogger operations • Task A o Create the keylogger functionality o Run in the background o Log keystrokes to keylog.txt. o Add a timestamp at the start of each session. • Task B o Implement command-line arguments to start the keylogger and specify the log file.
Nouf Hasya Ilma Binti Badaruddin (78334)	• Integrate the menu system for all operations. • Implement error checking for system calls in all tasks • Handle invalid inputs gracefully in both menu-based and command-line modes • Ensure the program prints success/failure messages for all operations • Conduct final testing and debugging for both Task A and Task B • Create the man page for the program and ensuring the overall structure is cohesive.

4.3 GROUP DISCUSSION COMMUNICATION MEDIUM



SP4133 Project
Nouf UNIMAS, Raiyani🌷, Sheana✨, You

Sheana✨: **Report:** https://1drv.ms/w/c/3e955219ca57365c/EeebswOHaJ9LlUxc8CLiKeEBV2ZbCCTsF2_tNwYSwGdEnw?e=CLckJP **Canva...**

30/12/2024

Nouf UNIMAS
> Sheana✨
> Report: https://1drv.ms/w/c/3e955219ca57365c/EeebswOHaJ9LlUxc8CLiKeEBV2ZbCCTsF2_tNwYSwGdEnw?e=CLckJP...

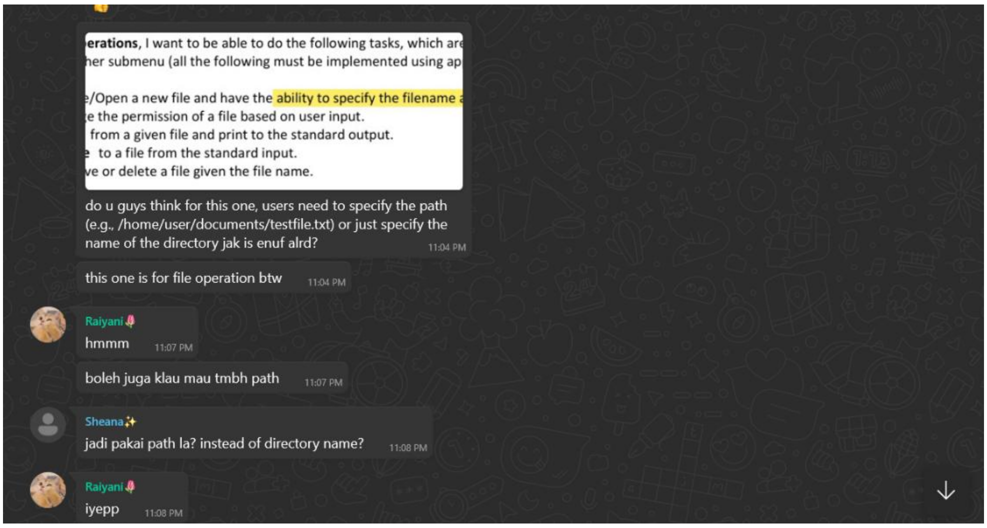
hii, for the report ada link lain tak? bila i tekan dia cakap i belum sign in but if nak sign in tak boleh, but if pergi home, logged in je to onedrive hmmmm (not in rush btw tanya jee) Edited 8:19 PM

Sheana✨
harrooo 8:22 PM
im out rn, balik later i send new linkk 8:22 PM

Nouf UNIMAS
okiee no rushhh thank youu 8:23 PM

Sheana✨
oke im back 9:37 PM
try this link 9:37 PM
https://1drv.ms/w/c/3e955219ca57365c/EeebswOHaJ9LlUxc8CLiKeEBV2ZbCCTsF2_tNwYSwGdEnw?e=zJL5hF 9:37 PM

Figure 38. Group Discussion Screenshot 1



erations, I want to be able to do the following tasks, which ar
her submenu (all the following must be implemented using ap

/Open a new file and have the ability to specify the filename
e the permission of a file based on user input.
from a given file and print to the standard output.
to a file from the standard input.
ve or delete a file given the file name.

do u guys think for this one, users need to specify the path (e.g., /home/user/documents/testfile.txt) or just specify the name of the directory jak is enuf alrd? 11:04 PM
this one is for file operation btw 11:04 PM

Raiyani🌷
hmmm 11:07 PM
boleh juga klau mau tmbh path 11:07 PM

Sheana✨
jadi pakai path la? instead of directory name? 11:08 PM

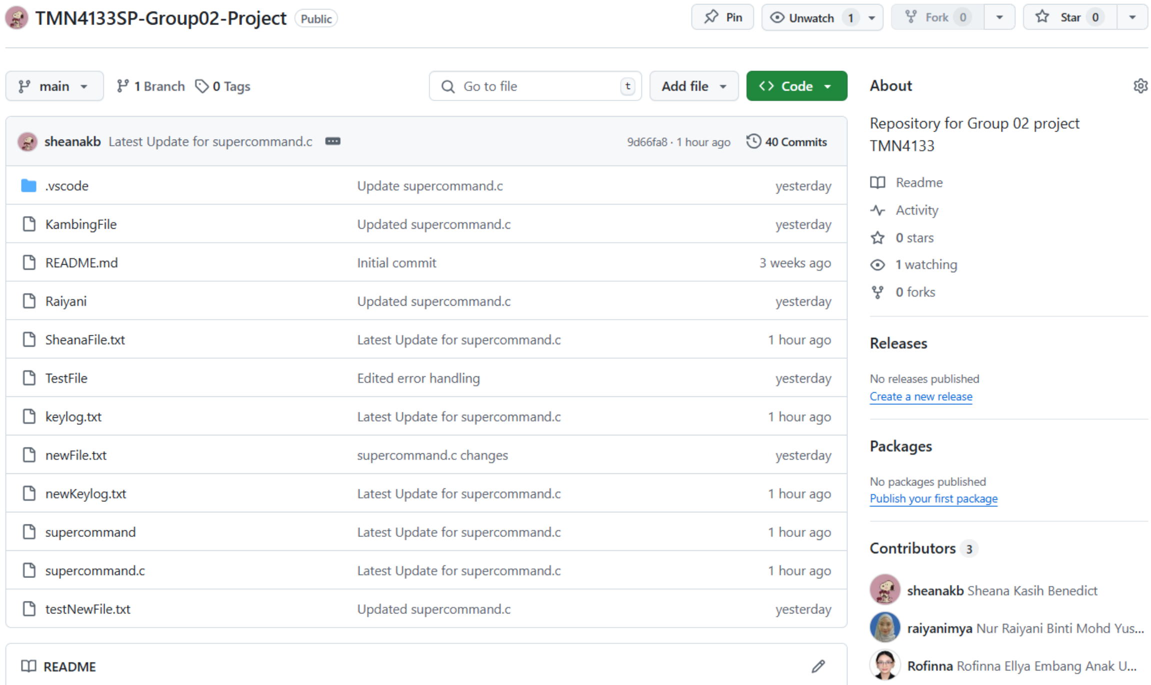
Raiyani🌷
iyepp 11:08 PM

Figure 39. Group Discussion Screenshot 2

In our opinion, using WhatsApp as a communication medium is one of the best methods to coordinate and communicate when it comes to group projects. We all use WhatsApp as our daily communication medium, which makes us more accessible and convenient for rea-time discussions. This platform not only enables file sharing but also streamlines communication due to our familiarity with all its features.

5.0 TASK D: USING GITHUB

Group 2 GitHub Link: https://github.com/sheanakb/TMN4133SP-Group02-Project



sheanakb Latest Update for supercommand.c 9d66fa8 · 1 hour ago 40 Commits

.vscode	Update supercommand.c	yesterday
KambingFile	Updated supercommand.c	yesterday
README.md	Initial commit	3 weeks ago
Raiyani	Updated supercommand.c	yesterday
SheanaFile.txt	Latest Update for supercommand.c	1 hour ago
TestFile	Edited error handling	yesterday
keylog.txt	Latest Update for supercommand.c	1 hour ago
newFile.txt	supercommand.c changes	yesterday
newKeylog.txt	Latest Update for supercommand.c	1 hour ago
supercommand	Latest Update for supercommand.c	1 hour ago
supercommand.c	Latest Update for supercommand.c	1 hour ago
testNewFile.txt	Updated supercommand.c	yesterday

README

About

Repository for Group 02 project TMN4133

Releases

No releases published
Create a new release

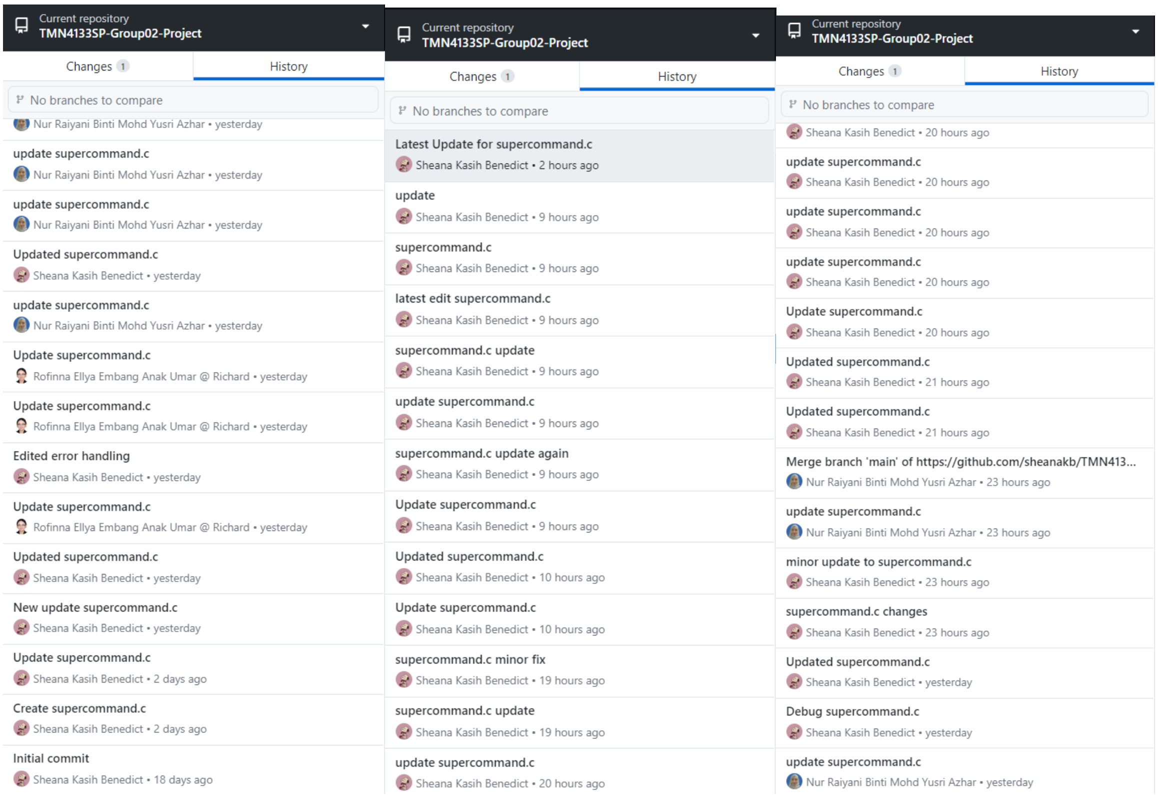
Packages

No packages published
Publish your first package

Contributors 3

sheanakb Sheana Kasih Benedict
raiyanimya Nur Raiyani Binti Mohd Yus...
Rofinna Rofinna Ellya Embang Anak U...

Figure 40: Group 2 GitHub Repository

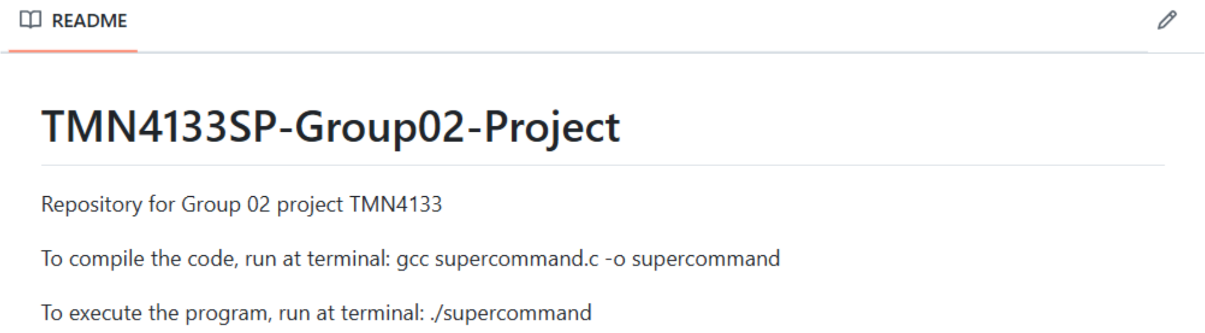


Nur Raiyani Binti Mohd Yusri Azhar • yesterday
update supercommand.c
Nur Raiyani Binti Mohd Yusri Azhar • yesterday
update supercommand.c
Nur Raiyani Binti Mohd Yusri Azhar • yesterday
Updated supercommand.c
Sheana Kasih Benedict • yesterday
update supercommand.c
Nur Raiyani Binti Mohd Yusri Azhar • yesterday
Update supercommand.c
Rofinna Ellya Embang Anak Umar @ Richard • yesterday
Update supercommand.c
Rofinna Ellya Embang Anak Umar @ Richard • yesterday
Edited error handling
Sheana Kasih Benedict • yesterday
Update supercommand.c
Rofinna Ellya Embang Anak Umar @ Richard • yesterday
Updated supercommand.c
Sheana Kasih Benedict • yesterday
New update supercommand.c
Sheana Kasih Benedict • yesterday
Update supercommand.c
Sheana Kasih Benedict • 2 days ago
Create supercommand.c
Sheana Kasih Benedict • 2 days ago
Initial commit
Sheana Kasih Benedict • 18 days ago

Latest Update for supercommand.c
Sheana Kasih Benedict • 2 hours ago
update
Sheana Kasih Benedict • 9 hours ago
supercommand.c
Sheana Kasih Benedict • 9 hours ago
latest edit supercommand.c
Sheana Kasih Benedict • 9 hours ago
supercommand.c update
Sheana Kasih Benedict • 9 hours ago
update supercommand.c
Sheana Kasih Benedict • 9 hours ago
supercommand.c update again
Sheana Kasih Benedict • 9 hours ago
Update supercommand.c
Sheana Kasih Benedict • 9 hours ago
Updated supercommand.c
Sheana Kasih Benedict • 10 hours ago
Update supercommand.c
Sheana Kasih Benedict • 10 hours ago
supercommand.c minor fix
Sheana Kasih Benedict • 19 hours ago
supercommand.c update
Sheana Kasih Benedict • 19 hours ago
update supercommand.c
Sheana Kasih Benedict • 20 hours ago

Sheana Kasih Benedict • 20 hours ago
update supercommand.c
Sheana Kasih Benedict • 20 hours ago
update supercommand.c
Sheana Kasih Benedict • 20 hours ago
update supercommand.c
Sheana Kasih Benedict • 20 hours ago
Update supercommand.c
Sheana Kasih Benedict • 20 hours ago
Updated supercommand.c
Sheana Kasih Benedict • 21 hours ago
Updated supercommand.c
Sheana Kasih Benedict • 21 hours ago
Merge branch 'main' of https://github.com/sheanakb/TMN413...
Nur Raiyani Binti Mohd Yusri Azhar • 23 hours ago
update supercommand.c
Nur Raiyani Binti Mohd Yusri Azhar • 23 hours ago
minor update to supercommand.c
Sheana Kasih Benedict • 23 hours ago
supercommand.c changes
Sheana Kasih Benedict • 23 hours ago
Updated supercommand.c
Sheana Kasih Benedict • yesterday
Debug supercommand.c
Sheana Kasih Benedict • yesterday
update supercommand.c
Nur Raiyani Binti Mohd Yusri Azhar • yesterday

Figure 41: Group 2 GitHub Collaboration and commit history



TMN4133SP-Group02-Project

Repository for Group 02 project TMN4133

To compile the code, run at terminal: gcc supercommand.c -o supercommand

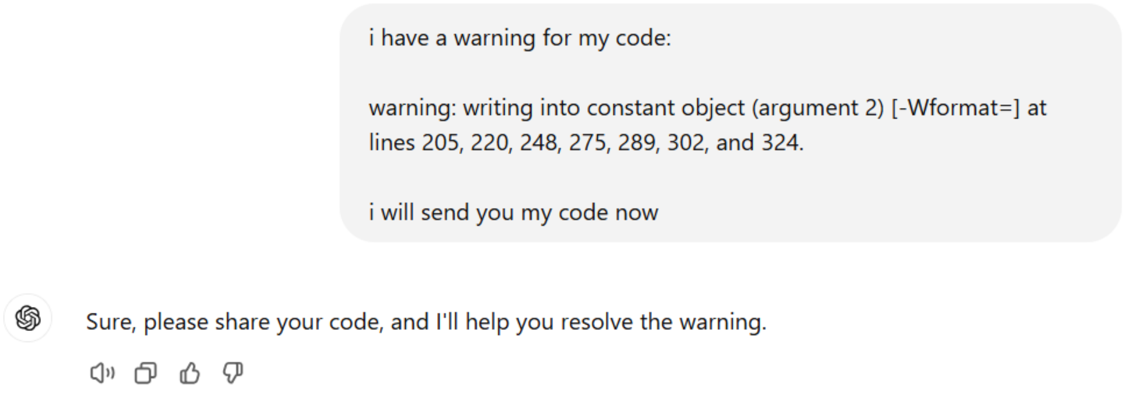
To execute the program, run at terminal: ./supercommand

Figure 41: Group 2 GitHub README page

6.0 CONCLUSION

In conclusion, the purpose of this project was to simplify system-level procedures by incorporating keylogger ability, directory management, and file operations into one program. This program is flexible as well as convenient, accommodating plenty of user preferences with its menu-based and command-line features. This project emphasized the significance of collaboration and problem-solving skills. Operations have been improved with the assistance of GitHub and meticulous testing to make sure of the program's accuracy. The project was an invaluable learning opportunity in every way, strengthening our programming knowledge and increasing our ability in facing more challenging tasks in the future.

APPENDIX

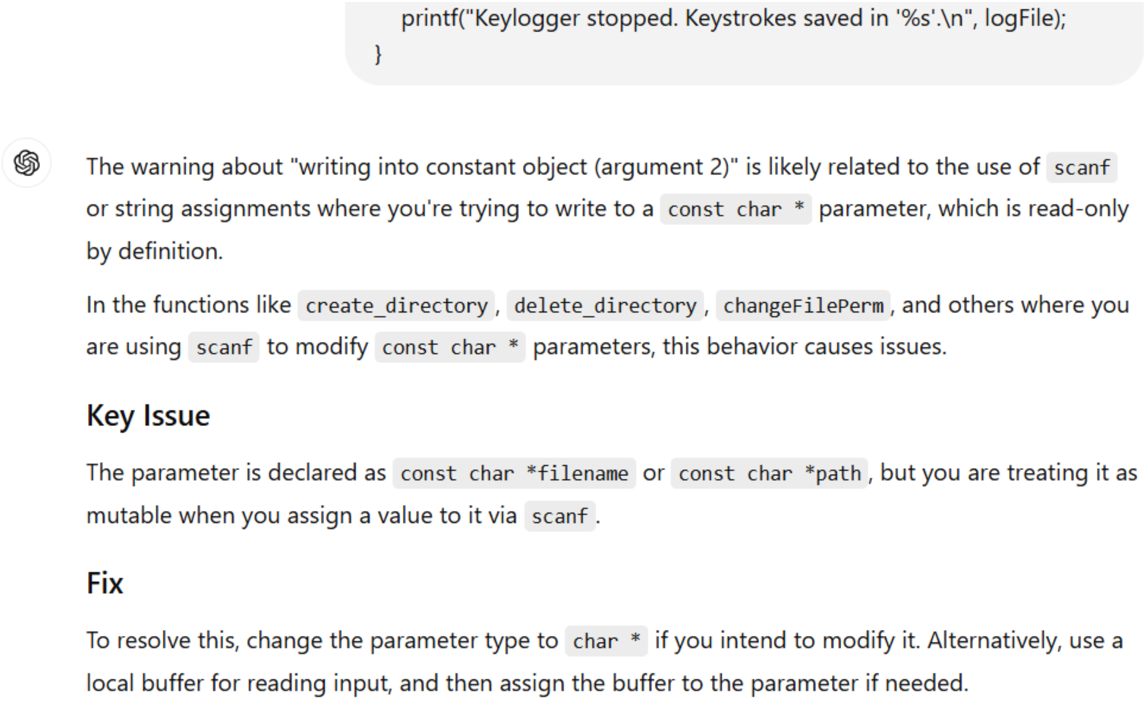


i have a warning for my code:

warning: writing into constant object (argument 2) [-Wformat=] at lines 205, 220, 248, 275, 289, 302, and 324.

i will send you my code now

Sure, please share your code, and I'll help you resolve the warning.



```
    printf("Keylogger stopped. Keystrokes saved in '%s'.\n", logFile);
}
```

The warning about "writing into constant object (argument 2)" is likely related to the use of `scanf` or string assignments where you're trying to write to a `const char *` parameter, which is read-only by definition.

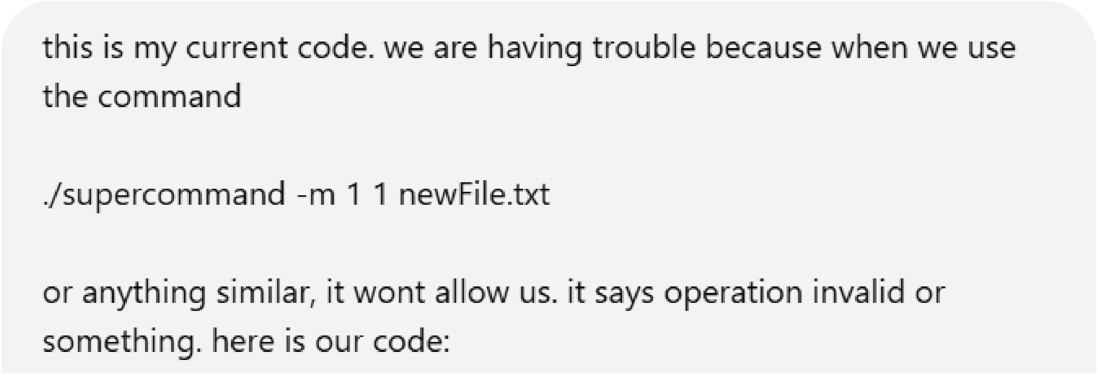
In the functions like `create_directory`, `delete_directory`, `changeFilePerm`, and others where you are using `scanf` to modify `const char *` parameters, this behavior causes issues.

Key Issue

The parameter is declared as `const char *filename` or `const char *path`, but you are treating it as mutable when you assign a value to it via `scanf`.

Fix

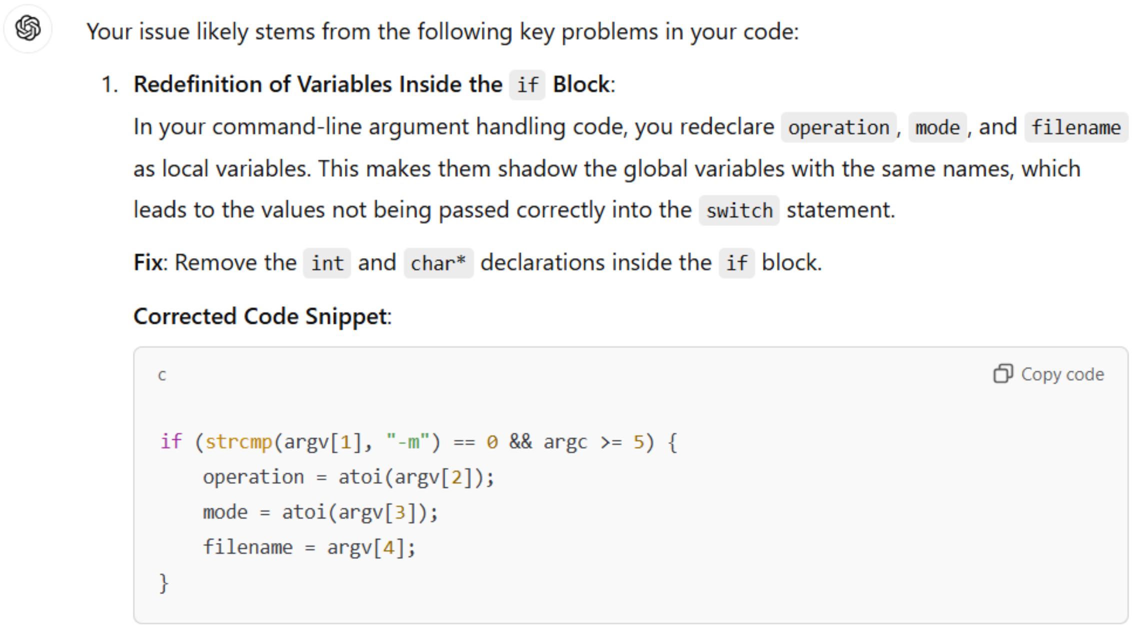
To resolve this, change the parameter type to `char *` if you intend to modify it. Alternatively, use a local buffer for reading input, and then assign the buffer to the parameter if needed.



this is my current code. we are having trouble because when we use the command

./supercommand -m 1 1 newFile.txt

or anything similar, it wont allow us. it says operation invalid or something. here is our code:



Your issue likely stems from the following key problems in your code:

1. **Redefinition of Variables Inside the `if` Block**:

   In your command-line argument handling code, you redeclare `operation`, `mode`, and `filename` as local variables. This makes them shadow the global variables with the same names, which leads to the values not being passed correctly into the `switch` statement.

   Fix: Remove the `int` and `char*` declarations inside the `if` block.

   Corrected Code Snippet:

```c
if (strcmp(argv[1], "-m") == 0 && argc >= 5) {
    operation = atoi(argv[2]);
    mode = atoi(argv[3]);
    filename = argv[4];
}
```