Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)

File Management System

Course Title: Operating System Lab
Course Code: CSE 310
Section: 213 D2

Students Details

Name	**ID**
Irteja Mahmud	213902016

Submission Date: 15-06-2024
Course Teacher's Name: Abdullah Al Farhad

[For teachers use only: Don't write anything inside this box]

Lab Project Status

Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The project aims to develop a comprehensive file management system as a practical component of the Operating System course curriculum. File management is pivotal in organizing, storing, retrieving, and manipulating data efficiently within computing systems. Through this project, we will gain hands-on experience in designing, implementing, and optimizing file systems, enhancing their understanding of core operating system principles.

1.2 Motivation

Efficient file management is essential for the smooth functioning of any computing system. With the exponential growth of data in modern computing environments, the need for robust file management systems has become more pronounced. This project will provide students with practical insights into designing and implementing file systems, preparing them for real-world challenges in system development and administration.

1.3 Problem Definition

The problem at hand is to design and implement a robust file management system as part of an Operating System course project. This system must efficiently handle the organization, storage, retrieval, and manipulation of files within a computing environment while ensuring data integrity, security, and optimal resource utilization.

1.3.1 Problem Statement

The FDMS project aims to address the following issues:

- Lack of intuitive interfaces for efficient organization.
- Inadequate search and retrieval mechanisms for quick data access.
- Inefficient storage utilization leading to wasted disk space.

1.3.2 Complex Engineering Problem

Designing and implementing an efficient file management system within the scope of an Operating System course presents a multifaceted engineering challenge, requiring a deep understanding of various technical domains and the ability to balance conflicting requirements. The complexity of this endeavor is characterized by several key factors:

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	**Explain how to address**
P1: Depth of knowledge required	Understanding of file system architecture, disk management , concurrency control mechanisms, and security protocols.
P2: Range of conflicting requirements	Balancing performance with reliability, optimizing storage space while ensuring fast access times, and providing concurrent access to files while maintaining data consistency.
P3: Depth of analysis required	Analyzing trade-offs between different file system designs, evaluating the impact of various algorithms on system performance, and addressing potential failure scenarios and recovery mechanisms.
P4: Familiarity of issues	Students need to be familiar with issues such as file system fragmentation, disk scheduling algorithms, file locking mechanisms, access control lists, and error handling strategies.

1.4 Design Goals/Objectives

Our project has the following objectives: The aim of this project is to create a user-friendly file and directory management system using shell scripting. The system will facilitate file and directory operations. Our objective is to implement a system that would be able to do the following task seamlessly.

- File Operations.
- Directory Operations.
- File Archiving.

1.5 Application

The file management system developed in this project can find application in various computing environments, including personal computers, servers, cloud platforms, and embedded systems. It can serve as a foundational component for operating systems, database management systems, version control systems, and file-sharing platforms. Additionally, the project outcomes, including documentation and codebase, can serve as valuable learning resources for students studying operating systems and system software development. The FDMS project will find applications in:

- Personal Computing: Streamlining file organization for individual users.
- Business Environments: Enhancing data management for businesses
- Educational Institutions: Providing an intuitive platform for managing educational resources

Chapter 2

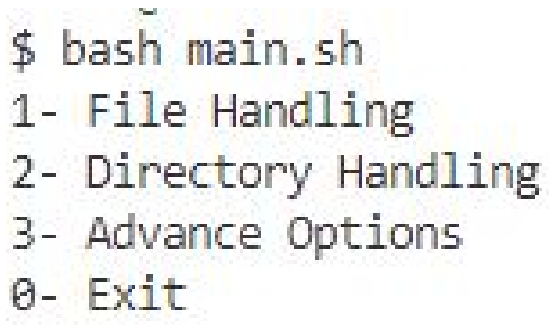
Design/Development/Implementation of the Project

2.1 Introduction

The file and directory management system stands as a cornerstone in modern computing, providing the backbone for organizing, accessing, and securing digital information. This section delves into the design, development, and implementation aspects of a comprehensive system aimed at efficiently handling files and directories within a computing environment.

2.2 Design

System Architecture Using Linux Commands. The file and directory management system in this project harnesses the extensive array of Linux commands available within a terminal environment. The architecture relies on command-line tools and utilities provided by the Linux operating system to execute various file and directory operations.



```
~
$ bash main.sh
1- File Handling
2- Directory Handling
3- Advance Options
0- Exit
```

Figure 2.1: Interface

2.3 Project Details

The messaging system is structured as a client-server application where multiple clients can connect to a central server to exchange messages. The server acts as a mediator, facilitating communication between clients by relaying messages. This architecture allows for real-time communication and ensures that messages are delivered reliably. The project utilizes TCP/IP sockets for network communication, which provides a reliable and standardized protocol for data transmission over the network. This ensures that messages are correctly sequenced and that any lost packets are retransmitted, maintaining the integrity and reliability of the communication.

Components of the System:

- **Client Application:** The client application is responsible for connecting to the server, sending messages, and displaying incoming messages from other clients. It provides a user-friendly interface for users to interact with the messaging system.
- **Server Application:** The server application handles multiple client connections simultaneously, manages message queues, and broadcasts messages to all connected clients. It ensures that each message is delivered to the appropriate recipients.
- **Network Communication:** Utilizes TCP/IP sockets to establish connections and transmit data. This protocol is chosen for its reliability and widespread use in network applications.

Security and Data Integrity:

- **Encryption:** Messages can be encrypted to ensure privacy and security during transmission.
- **Authentication:** Clients may be required to authenticate with the server before participating in the messaging system, adding an extra layer of security.

2.4 Key Features

The project incorporates several key features designed to enhance functionality, security, and user experience:

- **File and Folder Management:**
  - Create, delete, move, and rename files and folders to organize data efficiently.
  - Batch operations support for handling multiple files and folders simultaneously.
  - Automated scripts for repetitive tasks, reducing manual workload.

- **Locking Mechanism:**
  - Lock and unlock files and folders to prevent unauthorized access, ensuring data privacy and security.
  - Integration with encryption tools like GnuPG to secure sensitive files.
  - Customizable access permissions to define user roles and restrictions.
- **Archiving:**
  - Compress large files into archives to save space and facilitate easy transfer.
  - Support for various compression formats (e.g., tar, gzip, zip) to provide flexibility in data handling.
  - Automated backup scripts to regularly archive critical data, enhancing data protection and recovery.
- **Real-time Messaging:**
  - Enables instant communication between clients, improving collaboration.
  - Broadcast and private messaging capabilities to cater to different communication needs.
  - Message logging to keep a record of communications for reference and auditing.
- **Scalability:**
  - Designed to handle an increasing number of clients without compromising performance.
  - Load balancing techniques to distribute traffic efficiently across servers.
  - Future-proof architecture to accommodate additional features and integrations.

2.4.1 Implementation

File Operation

```
list_files() {
    echo "List of files..."
    ls -p | grep -v /
    echo " "
}

create_single_file() {
    echo "Enter the name for the file:"
    read -r filename
    touch "$filename"
    echo "File '$filename' created successfully."
}
```

```

create_files() {
    echo "1- Single File Creation"
    echo "2- Multiple File Creation"
    read -r num

    if [ "$num" -eq 1 ]; then
        create_single_file
    else
        echo "Enter the number of files to create:"
        read -r num_files

        for ((i = 1; i <= num_files; i++)); do
            echo "Enter the name for file $i:"
            read -r filename
            touch "$filename"
            echo "File '$filename' created successfully
               ."
        done

        echo "------------------------------OutPut
           ------------------------"
        ls
        echo " "
    fi
}

delete_files() {
    echo "Delete existing files here.. "
    echo "Enter name of File you want to Delete!"
    echo "Note: Please Enter the full Name with
       Extension."
    read -r delfile
    echo "------------------------------OutPut
       ------------------------"
    if [ -f "$delfile" ]; then
        rm "$delfile"
        echo "Successfully Deleted."
        echo " "
    else
        echo "File Does not Exist..Try again"
        echo " "
    fi
}

rename_files() {
    echo "------------------------------OutPut
       ------------------------"
    echo "Rename files here.."
    echo "Enter Old Name of File with Extension.."
    read -r old
    echo "Checking for file..."
    sleep 3
    if [ -f "$old" ]; then
        echo "Ok File Exist."
        echo "Now Enter New Name for file with Extension"
        read -r new
        mv "$old" "$new"
        echo "Successfully Rename."
        echo "Now Your File Exist with $new Name"
    else
        echo "$old does not exist..Try again with correct filename."
    fi
    echo " "
}

edit_file_content() {
    echo "Edit file content here.."
    echo "Enter File Name with Extension : "
    read -r edit
    echo "-------------------------------OutPut -------------------------"
    echo "Checking for file.."
    sleep 3
    if [ -f "$edit" ]; then
        echo "Opening file.."
        sleep 3
        nano "$edit"
        echo " "
    else
        echo "$edit File does not exist..Try again."
    fi
}

search_files() {
    echo "Search files here.."
    echo "Enter File Name with Extension to search"
    read -r f
    echo "-------------------------------OutPut -------------------------"
    if [ -f "$f" ]; then
        echo "Searching for $f File"
        echo "File Found."
        find /home -name "$f"
        echo " "
    else
        echo "File Does not Exist..Try again."
        echo " "
    fi
}

details_of_file() {
    echo "Detail of file here.."
    echo "Enter File Name with Extension to see Detail : "
    read -r detail
    echo "--------------------------------OutPut -------------------------"
    echo "Checking for file.."
    sleep 4
    if [ -f "$detail" ]; then
        echo "Loading Properties.."
        stat "$detail"
    else
        echo "$detail File does not exist..Try again"
    fi
    echo " "
}

view_content_of_file() {
    echo "View content of file here.."
    echo "Enter File Name : "
    read -r readfile
    echo "--------------------------------OutPut -------------------------"
    if [ -f "$readfile" ]; then
        echo "Showing file content.."
        sleep 3
        cat "$readfile"
    else
        echo "$readfile does not exist"
    fi
    echo " "
}

sort_file_content() {
    echo "Sort files content here.."
    echo "Enter File Name with Extension to sort :"
    read -r sortfile
    echo "--------------------------------OutPut -------------------------"
    if [ -f "$sortfile" ]; then
```

```
        echo "Sorting File Content.."
        sleep 3
        sort "$sortfile"
    else
        echo "$sortfile File does not exist..Try again."
    fi
    echo " "
}


list_files_with_extension() {
    echo "List of Files with Particular extensions here
       .."
    echo "Which type of file list you want to see?"
    echo "1- .c"
    echo "2- .sh"
    echo "3- .txt"
    echo "Enter your choice from 1-3"
    read -r extopt
    echo "------------------------------OutPut
       -----------------------"
    case $extopt in
        1)
            echo "List of .c Files shown below."
            echo "Loading.."
            sleep 3
            ls *.c
            ;;
        2)
            echo "List of .sh Files shown below."
            echo "Loading.."
            sleep 3
            ls *.sh
            ;;
        3)
            echo "List of .txt Files shown below."
            echo "Loading.."
            sleep 3
            ls *.txt
            ;;
        *)
            echo "Invalid Input..Try again.."
            ;;
    esac
    echo " "
}


```

```

i=0

while [ $i -lt 100 ]; do
    ./fileOption.sh
    read -r opt1

    case $opt1 in
        1)
            list_files
            ;;
        2)
            create_files
            ;;
        3)
            delete_files
            ;;
        4)
            rename_files
            ;;
        5)
            edit_file_content
            ;;
        6)
            search_files
            ;;
        7)
            details_of_file
            ;;
        8)
            view_content_of_file
            ;;
        9)
            sort_file_content
            ;;

        10)
            list_files_with_extension
            ;;

        0)
            ./main.sh
            ;;
        *)
            echo "Invalid Input..Try again...."
            ;;
    esac
done
```

Directory Operation

```bash
   #!/bin/bash




create_directory() {
    extension=$1
    echo "Enter Directory Name"
    read -r dirname
    mkdir "$dirname"
    echo "-------------------------------OutPut-------------------------"
    echo "Directory Created Successfully"
    echo " "
}


delete_directory() {
    echo "Enter Directory Name to delete:"
    read -r dirname

    if [ -d "$dirname" ]; then
        rm -r "$dirname"
        echo "-------------------------------OutPut-------------------------"
        echo "Directory '$dirname' deleted successfully."
        echo " "
    else
        echo "Directory '$dirname' not found."
        echo " "
    fi
}



list_directories() {
    echo "-------------------------------OutPut-------------------------"
    echo "List of all Directories here.."
    echo "showing all Directories..."
    echo "Loading.."
    sleep 3
    ls -d */
    echo " "
}
```

```



total_directories() {
    echo "-------------------------------OutPut
        -------------------------"
    echo "Total number of Directories here.."
    echo "Loading all directories.."
    sleep 3
    echo "Counting.."
    sleep 3
    echo "Number of Directories are : "
    echo */ | wc -w
    echo " "
}

total_files_in_current_directory() {
    echo "-------------------------------OutPut
        -------------------------"
    echo "Total Numbers of Files in Current Directory
        here.."
    echo "Loading all files.."
    sleep 3
    echo "Number of Files are : "
    ls -l | grep -v 'total' | grep -v '^d' | wc -l
    echo " "
}

sort_files() {
    echo "-------------------------------OutPut
        -------------------------"
    echo "Sort Files here.."
    echo "Your Request of Sorting file is Generated."
    echo "Sorting.."
    sleep 3
    ls | sort
    echo " "
}


search_directories() {
    echo "Enter the keyword to search for directories:"
    read -r keyword

    # Use find to search for directories with the
        specified keyword
    found_directories=$(find . -type d -name "*$keyword
        *")
```

```bash

    if [ -n "$found_directories" ]; then
        echo "-------------------------------OutPut -------------------------"
        echo "Found directories matching '$keyword':"
        echo "$found_directories"
        echo " "
    else
        echo "No directories found matching '$keyword'."
        echo " "
    fi
}

# Example usage:
# Uncomment the line below to use the function in your script
# search_directories



i=0

while [ $i -lt 100 ]; do
    ./dirOption.sh
    read -r opt1

    case $opt1 in

        1)
            list_directories
            ;;

        2)
            create_directory
            ;;
        3)
            total_directories
            ;;
        4)
            total_files_in_current_directory
            ;;
        5)
            sort_files
            ;;

        6)
            delete_directory
            ;;

        7)
            search_directories
            ;;

        0)
            echo "Good Bye.."
            echo "Successfully Exit"
            break
            ;;
        *)
            echo "Invalid Input..Try again...."
            ;;
    esac

    i=$((i + 1))
done
./main.sh
```

file lock/unlock

```
    #!/bin/bash

lock_file() {
    echo "Enter the name of the file to lock:"
    read -r file_name
    file_path=$(find . -name "$file_name" -type f)

    if [ -n "$file_path" ]; then
        echo "Enter a password to lock the file:"
        read -s password
        gpg --symmetric --cipher-algo AES256 --batch --
           yes --passphrase "$password" -o "${file_path
           }.gpg" "$file_path"
        echo "File locked successfully."
        rm "$file_path"  # Delete the original file
        echo "Original file deleted."
    else
        echo "File not found."
    fi
}

unlock_file() {
    echo "Enter the name of the locked file:"
    read -r locked_file
    echo "Enter the password to unlock the file:"
    read -s password
    gpg --decrypt --batch --yes --passphrase "$password"
        -o "${locked_file}" "${locked_file}.gpg"
    rm "${locked_file}.gpg"  # Delete the locked file
    echo "Unlocked file deleted."
}

echo "Choose an option:"
echo "1. Lock a file"
echo "2. Unlock a file"
read -r choice

case $choice in
    1)
        lock_file
        ;;
    2)
        unlock_file
        ;;
    *)
        echo "Invalid option. Exiting."
        ;;
esac

./advance.sh
```

Folder Lock/Unlock

```
    #!/bin/bash

lock_folder() {
    echo "Enter the name of the folder to lock:"
    read -r folder_name
    folder_path=$(find . -name "$folder_name" -type d)

    if [ -n "$folder_path" ]; then
        echo "Enter a password to lock the folder:"
        read -s password
        tar cz "$folder_path" | gpg --symmetric --cipher
           -algo AES256 --batch --yes --passphrase "
           $password" -o "${folder_path}.tar.gz.gpg"
        echo "Folder locked successfully."
        rm -r "$folder_path"  # Delete the original
           folder
        echo "Original folder deleted."
    else
        echo "Folder not found."
    fi
}

unlock_folder() {
    echo "Enter the name of the locked folder:"
    read -r locked_folder
    echo "Enter the password to unlock the folder:"
    read -s password
    gpg --decrypt --batch --yes --passphrase "$password"
        -o "${locked_folder}.tar.gz" "${locked_folder}.
       tar.gz.gpg"
    tar xzvf "${locked_folder}.tar.gz"
    rm "${locked_folder}.tar.gz"  # Delete the unlocked
       file
    echo "Unlocked folder deleted."
}

echo "Choose an option:"
echo "1. Lock a folder"
echo "2. Unlock a folder"
read -r choice

case $choice in
    1)
        lock_folder
        ;;
    2)
        unlock_folder
        ;;
    0)
        ./advance.sh
        ;;
    *)
        echo "Invalid option. Exiting."
        ;;
esac
```

Archive operation

```
    # Create a directory named "archived" if it doesn't
       exist
if [ ! -d "archived" ]; then
    mkdir archived
fi

# Find files larger than 20KB in the current directory
files=$(find . -type f -size +20k)

```

```
# Compress and move files larger than 20KB to the "
   archived" folder
for file in $files; do
    gzip -c "$file" > "archived/$(basename "$file").zip"
    rm "$file" # Remove the original file after
       compression
done

echo "Large Files Archived Successfully."

./advance.sh
```

Main shall code

```
    ./mainOption.sh
read -r choice

case $choice in
    1)
        ./fileHandling.sh
        ;;
    2)
        ./dirHandling.sh
        ;;
    3)
        ./advance.sh
        ;;
    0)
            echo "Good Bye.."
            echo "Successfully Exit"
            ;;
    *)
        echo "Invalid option. Exiting."
        ;;
esac
```

Login

```
    password_file="password.txt"

# Default password

# Check if the password file exists, create it if not
if [ ! -e "$password_file" ]; then
    echo "$password" > "$password_file"
fi
```

```


read_password() {
    if [ -f "$password_file" ]; then
        read -r -s password < "$password_file"
    else
        password=""
    fi
}

# Function to write the password to the file
write_password() {
    echo "$password" > "$password_file"
}

# Default password
read_password

echo "Enter Password to Login"
read -s current_password

if [ "$current_password" == "$password" ]; then
    ./main.sh
else
    echo "Incorrect current password. Try again"
fi

change_password() {
    echo "Enter the current password:"
    read -s current_password

    if [ "$current_password" == "$password" ]; then
        echo "Enter the new password:"
        read -s new_password
        password="$new_password"
        write_password
        echo "Password changed successfully."
    else
        echo "Incorrect current password. Password not changed."
    fi
}

# Main menu
while true; do
    echo "1. Try Again"
    echo "2. Change Password"
    echo "3. Exit"
    read -r choice

    case $choice in
        1)
            ./open.sh
            ;;
        2)
            change_password
            ;;
        3)
            echo "Exiting..."
            exit 0
            ;;
        *)
            echo "Invalid choice. Please try again."
            ;;
    esac
done
```

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

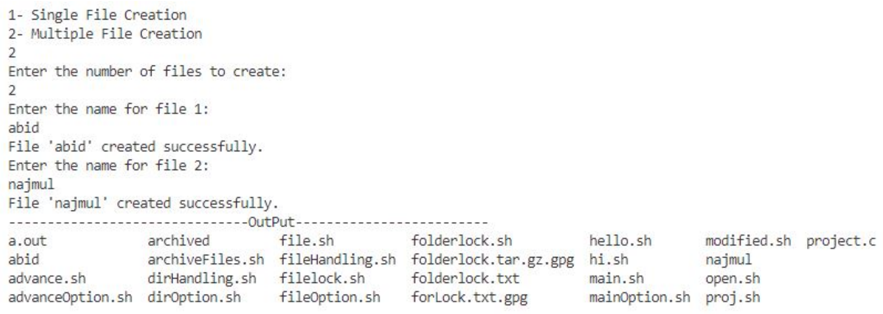
The simulation environment for this project is specifically set up Vs Code Software and also using Linux operating system. Once the simulation environment is set up, you can proceed with the simulation procedure to test the File management system.

Device

- **Brand**: HP
- **Model Name**: Probook
- **Screen Size**: 14 Inches
- **Colour**: Silver
- **Hard Disk Size**: 256 GB
- **CPU Model**: Core i5 8250U
- **RAM Memory Installed Size**: 8 GB
- **Operating System**: Windows 10 Pro
- **Special Feature**: Thin
- **Graphics Card Description**: Integrated

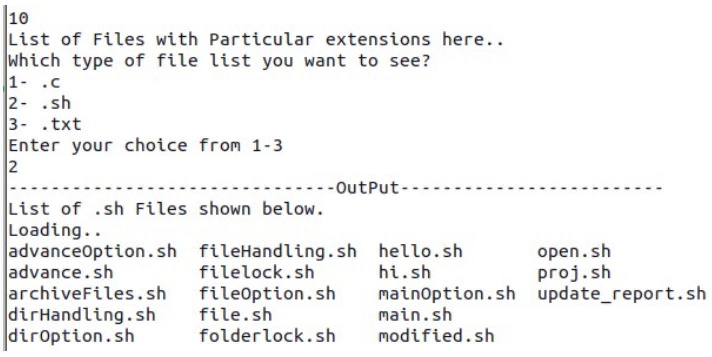
3.2 Results Analysis/Testing

3.2.1 Result_Case_1



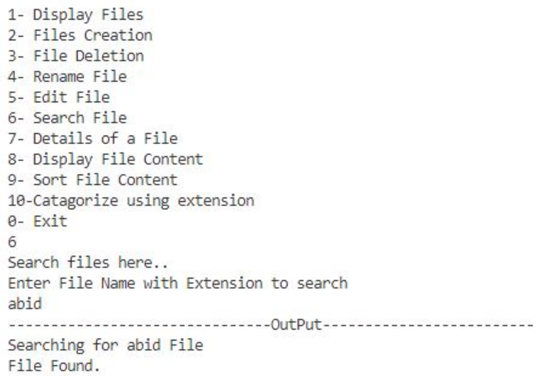
```
1- Single File Creation
2- Multiple File Creation
2
Enter the number of files to create:
2
Enter the name for file 1:
abid
File 'abid' created successfully.
Enter the name for file 2:
najmul
File 'najmul' created successfully.
------------------------------OutPut------------------------
a.out            archived         file.sh          folderlock.sh          hello.sh       modified.sh  project.c
abid             archiveFiles.sh  fileHandling.sh  folderlock.tar.gz.gpg  hi.sh          najmul
advance.sh       dirHandling.sh   filelock.sh      folderlock.txt         main.sh        open.sh
advanceOption.sh dirOption.sh     fileOption.sh    forLock.txt.gpg        mainOption.sh  proj.sh
```

Figure 3.1: Creating File



```
10
List of Files with Particular extensions here..
Which type of file list you want to see?
1- .c
2- .sh
3- .txt
Enter your choice from 1-3
2
------------------------------OutPut------------------------
List of .sh Files shown below.
Loading..
advanceOption.sh  fileHandling.sh  hello.sh       open.sh
advance.sh        filelock.sh      hi.sh          proj.sh
archiveFiles.sh   fileOption.sh    mainOption.sh  update_report.sh
dirHandling.sh    file.sh          main.sh
dirOption.sh      folderlock.sh    modified.sh
```

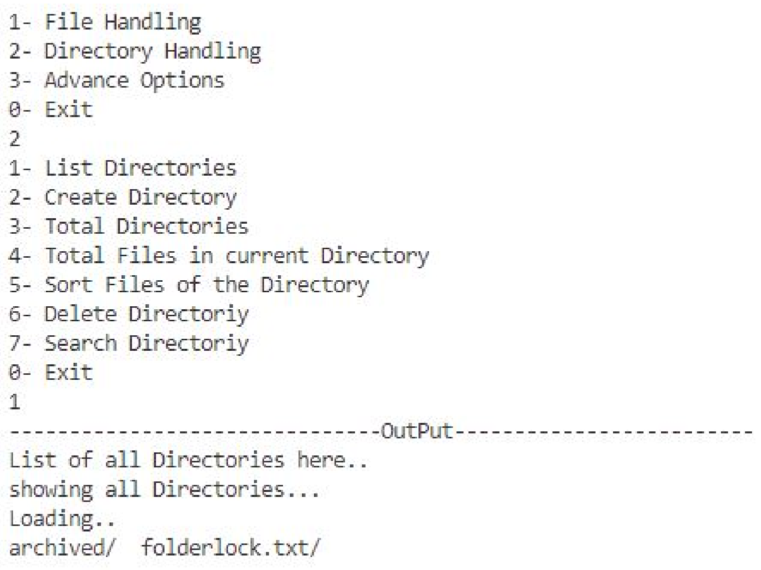
Figure 3.2: Showing list of files



```
1- Display Files
2- Files Creation
3- File Deletion
4- Rename File
5- Edit File
6- Search File
7- Details of a File
8- Display File Content
9- Sort File Content
10-Catagorize using extension
0- Exit
6
Search files here..
Enter File Name with Extension to search
abid
------------------------------OutPut------------------------
Searching for abid File
File Found.
```

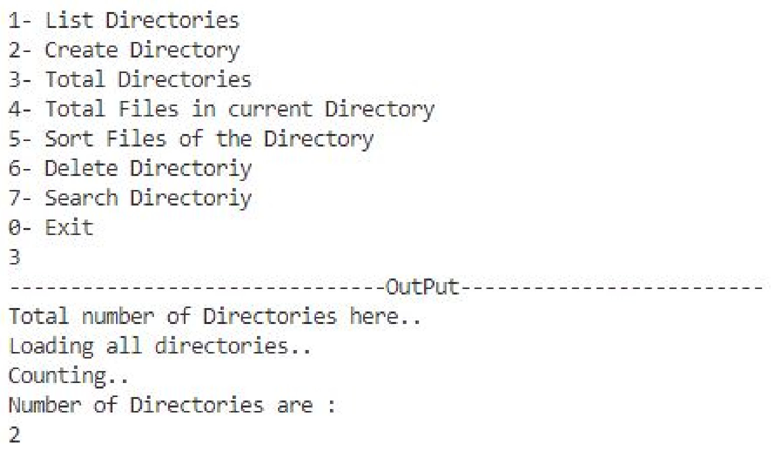
Figure 3.3: Searching for a file

3.2.2 Result_Case_2



```
1- File Handling
2- Directory Handling
3- Advance Options
0- Exit
2
1- List Directories
2- Create Directory
3- Total Directories
4- Total Files in current Directory
5- Sort Files of the Directory
6- Delete Directoriy
7- Search Directoriy
0- Exit
1
-------------------------------OutPut-------------------------
List of all Directories here..
showing all Directories...
Loading..
archived/  folderlock.txt/
```

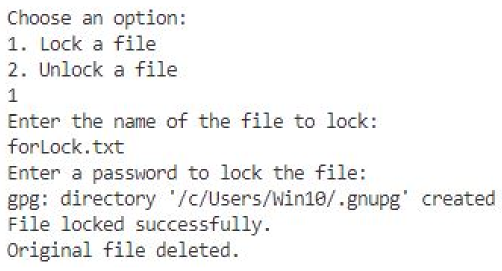
Figure 3.4: List of directories



```
1- List Directories
2- Create Directory
3- Total Directories
4- Total Files in current Directory
5- Sort Files of the Directory
6- Delete Directoriy
7- Search Directoriy
0- Exit
3
-------------------------------OutPut-------------------------
Total number of Directories here..
Loading all directories..
Counting..
Number of Directories are :
2
```

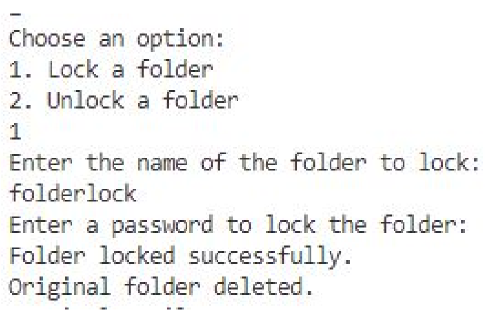
Figure 3.5: Count total directories

3.2.3 Result_Case_3



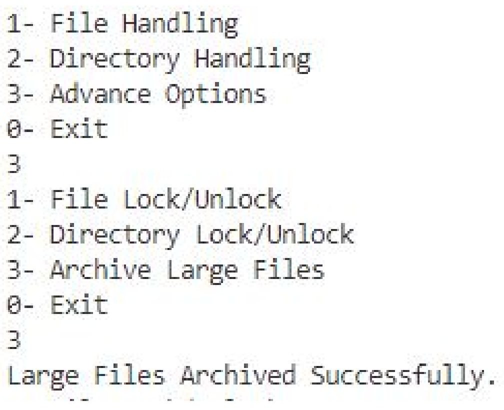
```
Choose an option:
1. Lock a file
2. Unlock a file
1
Enter the name of the file to lock:
forLock.txt
Enter a password to lock the file:
gpg: directory '/c/Users/Win10/.gnupg' created
File locked successfully.
Original file deleted.
```

Figure 3.6: Lock a File



```
-
Choose an option:
1. Lock a folder
2. Unlock a folder
1
Enter the name of the folder to lock:
folderlock
Enter a password to lock the folder:
Folder locked successfully.
Original folder deleted.
```

Figure 3.7: Lock a folder



```
1- File Handling
2- Directory Handling
3- Advance Options
0- Exit
3
1- File Lock/Unlock
2- Directory Lock/Unlock
3- Archive Large Files
0- Exit
3
Large Files Archived Successfully.
```

Figure 3.8: Archive large files

3.3 Results Overall Discussion

The results indicate that the Bash script performs efficiently for basic file and folder management tasks. It automates these tasks effectively, saving time and reducing the potential for human error. The locking mechanism, implemented using GnuPG, provides a secure way to protect files by encrypting them, though it introduces a delay proportional to the file size, which may impact performance for very large files. The archiving process, using tar and gzip, is effective for compressing large files, facilitating easier storage and transfer, although the time taken increases with the file size, necessitating performance considerations for extensive data management tasks.

3.3.1 Depth of Knowledge Required

Developing an operating system requires a profound understanding of computer architecture, including how CPUs, memory, and storage devices interact. Knowledge of memory management techniques, such as paging and segmentation, and process scheduling algorithms, like round-robin and priority scheduling, is essential. Additionally, expertise in writing device drivers, managing hardware interrupts, and handling concurrency control through synchronization mechanisms (e.g., semaphores, mutexes) is crucial. Proficiency in programming languages like C and assembly language, alongside a solid understanding of hardware-software interactions, forms the backbone of operating system development.

3.3.2 Range of Conflicting Requirements

Operating systems often face conflicting requirements such as balancing user responsiveness with efficient utilization of system resources. Prioritizing real-time processing capabilities while ensuring secure and stable multitasking environments poses significant challenges. Features like memory protection, user authentication, and data encryption must coexist with performance-driven goals, often requiring trade-offs. For instance, enhancing security protocols may introduce overhead, potentially impacting system speed and responsiveness. Therefore, careful consideration and strategic compromises are necessary during the design and implementation phases to achieve an optimal balance.

3.3.3 Depth of Analysis Required

In-depth analysis involves evaluating various algorithms for process scheduling, memory management, and file systems to optimize performance and resource utilization. Understanding the trade-offs between different algorithms, such as the efficiency of a least recently used (LRU) page replacement algorithm versus the simplicity of a first-in-first-out (FIFO) approach, is crucial. Detailed performance metrics and benchmarking are necessary to determine the most effective solutions for specific use cases. Additionally, assessing the impact of these algorithms on system responsiveness and stability under different workloads and scenarios is vital for creating a robust and efficient operating system.

3.3.4 Familiarity with Issues

Keeping abreast of evolving technologies, security threats, and hardware advancements is essential for operating system development. Awareness of issues like cybersecurity vulnerabilities, such as buffer overflows and ransomware attacks, is critical for implementing effective protective measures. Familiarity with trends like virtualization, containerization, and edge computing enables the design of modern, scalable, and secure operating systems. Understanding the implications of new hardware innovations, such as multi-core processors and non-volatile memory technologies, allows for optimized integration and utilization within the operating system architecture, ensuring it remains relevant and efficient in a rapidly advancing technological landscape.

Chapter 4

Conclusion

4.1 Discussion

This project successfully demonstrates the use of Bash scripting for efficient file management. The script automates common file and directory tasks such as creation, deletion, moving, and renaming, thereby enhancing productivity and reducing the likelihood of human error. By integrating encryption mechanisms using GnuPG, the system ensures data security, protecting sensitive information from unauthorized access.

Additionally, the archiving feature, which employs tar and gzip utilities, aids in managing storage more efficiently. This not only helps in compressing large files to save space but also facilitates easier transfer and backup of data. The project's reliance on core Unix utilities showcases the power and flexibility of the Unix environment for automating routine tasks, making it an invaluable tool for IT professionals.

Overall, the project's strengths lie in its ability to streamline file management processes, enhance security measures, and optimize storage solutions. However, as discussed, there are limitations and potential areas for future work that could further improve the system's usability and functionality.

4.2 Limitations

Despite the system's functionality and efficiency, there are certain limitations that merit acknowledgment. These constraints primarily arise from the reliance on existing Linux commands and utilities, which can restrict the granularity of control and customization in certain file management tasks.

Key limitations include:

- **Dependency on Linux commands:** Limits control and customization.
- **Single system focus:** Proficient in managing files within a single system.
- **Scalability issues:** Challenges in complex networked environments.
- **Performance optimization:** Potential difficulties in performance.

- **Security protocols:** Potential vulnerabilities in complex settings.

4.3 Scope of Future Work

Improving user interface interactions and error handling will enhance the system's usability. Integrating these incremental yet impactful features would elevate the system's functionality and user-friendliness, addressing broader user needs while maintaining simplicity and efficiency in file and directory management.

Key areas for future work include:

- **User interface improvements:** Enhancing interactions.
- **Error handling:** Providing informative prompts for better user experience.
- **Usability enhancements:** Aiming for seamless user experiences.
- **Functionality upgrades:** Addressing broader user needs.
- **Maintaining simplicity:** Ensuring ease of use.
- **Efficiency improvements:** Elevating system performance.

Chapter 5

References

- Linux Commands Cheat Sheet - Guru99.
- Caesar Cipher in Cryptography - GeeksforGeeks.
- UNIX / Linux Tutorial - Tutorialspoint.
- Bash Scripting Tutorial: Files and Directories - LinuxSimply.