Simplified UNIX-Like Utilities

Submitted By:
Varshitha Masaram - CS22B1071
P. Naga Sripada - CS22B1018
N. Priyadarshini - CS22B1009
G. Satya Priya - CS22B1012

Date of Submission:
Nov 24, 2024

Contents

1 Objective

The objective of this project is to develop a series of lightweight and simplified versions of UNIX utilities. These utilities mimic common commands like `cat`, `ls`, `grep`, and others. The purpose is to gain hands-on experience with system programming by implementing these commands from scratch in C.

Additionally, the project involves:

- Developing a **custom shell** to execute the commands interactively.
- Writing a **Makefile** for efficient compilation and testing.
- Documenting the project with execution screenshots and detailed explanations.

2 Introduction

UNIX utilities are essential tools for file and directory management. By re-implementing them from scratch, we learn:

- System calls for file operations like `open`, `read`, `write`, and `close`.
- Directory traversal and metadata extraction using APIs like `opendir` and `stat`.
- String manipulation and pattern matching.
- Error handling and edge case management.

This document provides a detailed description of the implemented utilities, their functionalities, key concepts, and usage examples.

3 Setting Up the Environment and Creating Files

This section describes the steps to set up the environment for the custom UNIX utilities project, including creating directories, writing code files, and configuring the system path for easy execution of the utilities.

3.1 Creating a Directory

To organize the custom UNIX utility programs, a dedicated directory is created.

`Command Used:`

```
mkdir custom_unix_utilities
```

Explanation:

- The `mkdir` command creates a new directory named `custom_unix_utilities`.
- Once created, navigate into this directory using the `cd` command:

```
cd custom_unix_utilities
```

```
varshitha@Varshitha:~$
varshitha@Varshitha:~$ mkdir custom_unix_utilities
varshitha@Varshitha:~$ cd custom_unix_utilities
```

Figure 1: Creating directory

3.2 Creating Code Files Using `nano`

Code files for the custom utilities can be created and edited using the `nano` text editor.

`Command Used:`

```
nano custom_cat.c
```

Explanation:

- The `nano` command opens the `custom_cat.c` file in the terminal editor.
- Inside the editor:
  - Write the C code for the `custom_cat` utility.
  - Press `CTRL + O` to save the file and `CTRL + X` to exit.
- For other utilities, create files similarly. For example:

```
nano custom_ls.c
```

```
varshitha@Varshitha:~/custom_unix_utilities$ nano custom_ls.c
varshitha@Varshitha:~/custom_unix_utilities$
varshitha@Varshitha:~/custom_unix_utilities$
varshitha@Varshitha:~/custom_unix_utilities$ nano custom_cat.c
varshitha@Varshitha:~/custom_unix_utilities$
varshitha@Varshitha:~/custom_unix_utilities$ nano custom_grep.c
varshitha@Varshitha:~/custom_unix_utilities$ nano custom_wc.c
varshitha@Varshitha:~/custom_unix_utilities$ nano custom_cp.c
varshitha@Varshitha:~/custom_unix_utilities$ nano custom_mv.c
varshitha@Varshitha:~/custom_unix_utilities$ nano custom_rm.c
varshitha@Varshitha:~/custom_unix_utilities$ ls
custom_cat.c  custom_grep.c  custom_mv.c  custom_wc.c
custom_cp.c   custom_ls.c    custom_rm.c
```

Figure 2: Creating code files using nano

3.3 Setting Up Custom Commands with `export`

To run the custom utilities directly, the directory path is added to the `PATH` environment variable.

`Command Used:`

```
export PATH=$PATH:/home/varshitha/custom_unix_utilities
```

Explanation:

- The `export` command temporarily appends `/home/varshitha/custom_unix_utilities` to the `PATH`.
- This allows the utilities (e.g., `custom_cat`) to be executed without specifying the full path.
- Verify this by checking the updated `PATH` variable:

```
echo $PATH
```

```
varshitha@Varshitha:~/custom_unix_utilities$ pwd
/home/varshitha/custom_unix_utilities
varshitha@Varshitha:~/custom_unix_utilities$ export PATH=$PATH:/home/varshitha/custom_unix_utilities
varshitha@Varshitha:~/custom_unix_utilities$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib:/mnt/c/Program Files/Python312/Scripts/:/mnt/c/Program Files/Python312/:/mnt/c/Program Files/Common Files/Oracle/Java/javapath:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerShell/v1.0/:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/Program Files/dotnet/:/mnt/c/Program Files/Git/cmd:/mnt/c/Program Files/Git/mingw64/bin:/mnt/c/Program Files/Git/usr/bin:/mnt/c/MinGW/bin:/mnt/c/Program Files/MySQL/MySQL Server 8.0/bin:/mnt/c/Program Files/Java/jdk-22/bin:/mnt/c/Program Files/nodejs/:/mnt/c/ProgramData/chocolatey/bin:/mnt/c/Program Files/MySQL/MySQL Shell 8.0/bin/:/mnt/c/Users/varsh/AppData/Local/Microsoft/WindowsApps:/mnt/c/Program Files/Git/bin:/mnt/c/MinGW/bin:/mnt/c/Users/varsh/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Users/varsh/AppData/Local/Programs/MiKTeX/miktex/bin/x64/:/mnt/c/iverilog/bin:/mnt/c/iverilog/gtkwave/bin:/mnt/c/Users/varsh/AppData/Roaming/npm:/snap/bin:/home/user/custom_unix_utilities:/home/user/custom_unix_utilities:/home/user/custom_unix_utilities:/home/varshitha/custom_unix_utilities
```

3.4 Making Changes Persistent with `nano ~/.bashrc`

To make the `PATH` changes permanent, update the `~/.bashrc` file.

`Command Used:`

```
nano ~/.bashrc
```

Explanation:

- The `~/.bashrc` file is a shell script that initializes the terminal environment.
- Add the following line at the end of the file to make the `PATH` update permanent:

```
export PATH=$PATH:/home/varshitha/custom_unix_utilities
```

- Save the file and exit the editor.

3.5 Applying Changes with `source ~/.bashrc`

To reload the updated `~/.bashrc` file without restarting the terminal:

`Command Used:`

```
source ~/.bashrc
```

Explanation:

- The `source` command applies changes made to the `~/.bashrc` file.
- This ensures the updated `PATH` configuration is immediately available.

For ensuring permissions we used chmod:

```
varshitha@Varshitha:~/custom_unix_utilities$ nano ~/.bashrc
varshitha@Varshitha:~/custom_unix_utilities$ source ~/.bashrc
```

```
varshitha@Varshitha:~/custom_unix_utilities$ chmod +x custom_*
```

3.6 Verifying Configuration with `echo`

To confirm that the directory has been added to the `PATH` variable:

`Command Used:`

```
echo $PATH
```

Explanation:

- The `echo` command displays the current value of the `PATH` variable.
- Verify that `/home/varshitha/custom_unix_utilities` is listed.
- Example output:

```
/usr/local/bin:/usr/bin:/bin:/home/varshitha/custom_unix_utilities
```

3.7 Compiling and Executing Custom Utilities

After writing the C code, compile the utility and execute it:

`Command Used:`

```
gcc -o custom_cat custom_cat.c
custom_cat sample.txt
```

Explanation:

- The `gcc` command compiles the C code into an executable file.
- The compiled utility (`custom_cat`) is then executed with a file (`sample.txt`) as input.

4 Implemented Utilities

4.1 custom_ls

Description: Displays the contents of a directory, similar to the `ls` command.
Functionalities:

- Lists all files and subdirectories in the specified path.
- Displays file types (e.g., directories, regular files).
- Recursively lists subdirectory contents using the `-l` flag.
- Displays hidden files using the `-a` flag.

- Display files with its size in bytes `-s` flag

Key Concepts:

- **Directory Traversal:** Implemented using `opendir`, `readdir`, and `closedir`.
- **File Metadata:** Extracted using the `stat` system call.

Usage:

```
custom_ls [options] [path]
# Examples:
custom_ls               # List files in the current directory.
custom_ls -l            # Recursively list files.
custom_ls -a            # Show hidden files.
custom_ls -s    #List files with its size in bytes
```

4.2 Demonstration

The following figure illustrates how the

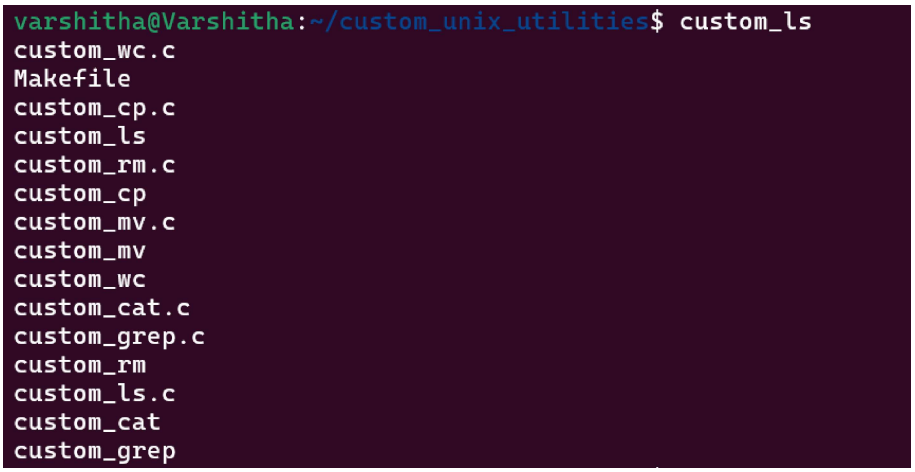

Figure 3: custom_ls demonstration

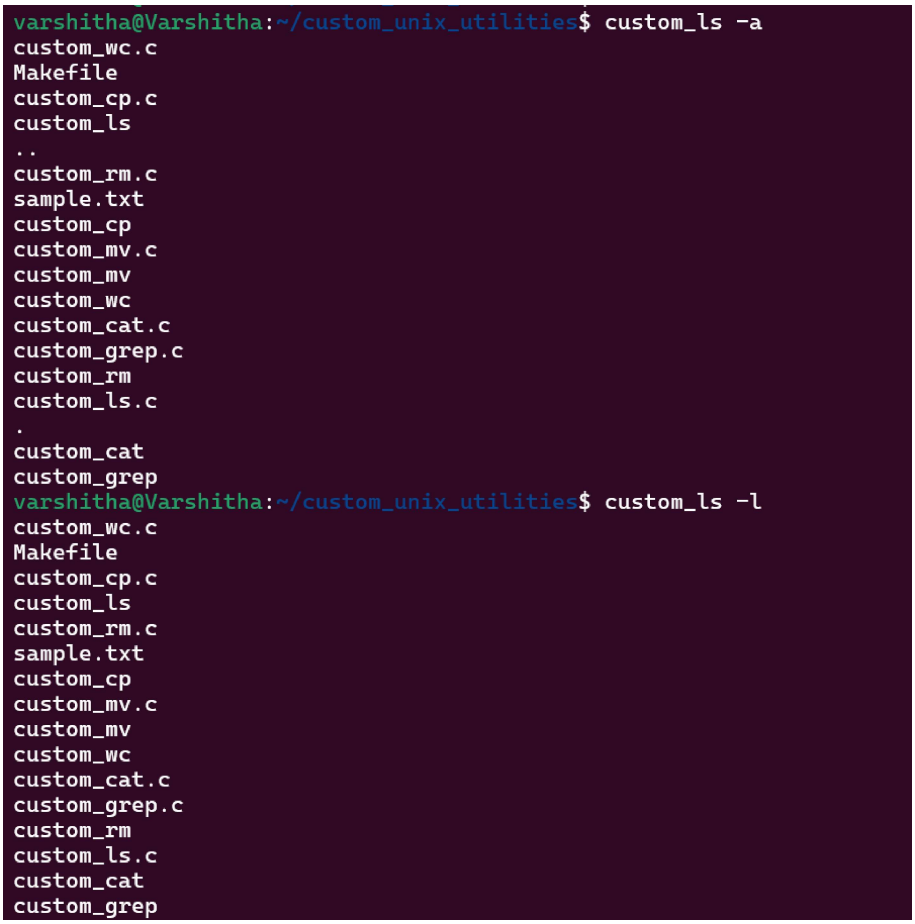

Figure 4: custom_ls -l,-a demonstration

Figure 5: custom_ls -s demonstration

4.3 custom_cat

Description: Reads and displays the contents of files, similar to the `cat` command.

Functionalities:

- Displays the contents of one or more files.
- Concatenates multiple files and displays them together.
- Provides options to display text in uppercase (`-u`) or lowercase (`-l`).

Key Concepts:

- **File Reading:** Implemented using `fopen` and `fgets`.
- **Text Manipulation:** Uses `toupper` and `tolower`.

Usage:

```
custom_cat [-u|-l] file1 file2 ...
# Examples:
custom_cat file1.txt                # Display file contents.
custom_cat -u file1.txt             # Display contents in uppercase.
custom_cat file1.txt file2.txt      # Concatenate and display multiple
    files.
```

4.4 Demonstration

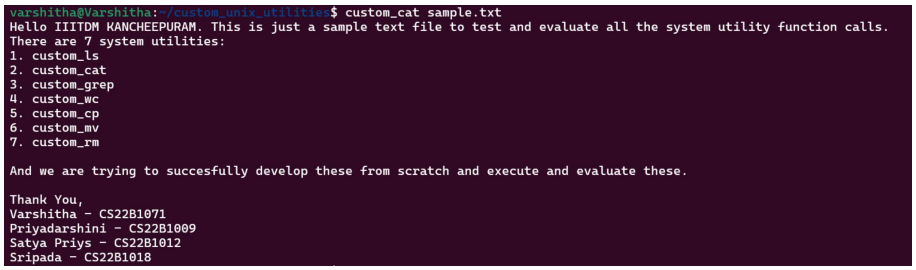

Figure 6: custom_cat demonstration

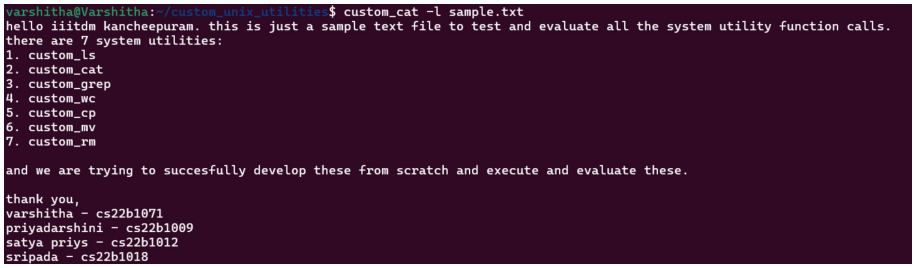

Figure 7: custom_cat -l demonstration

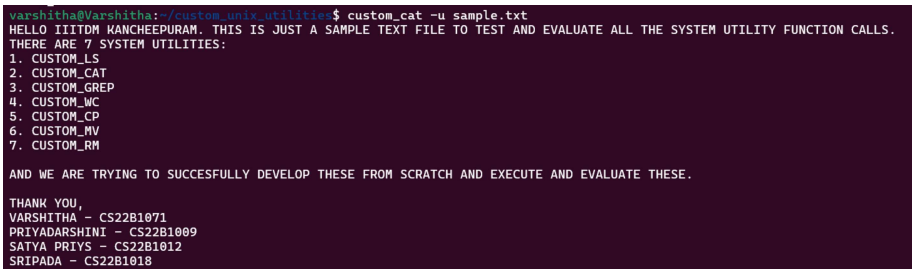

Figure 8: custom_cat -u demonstration

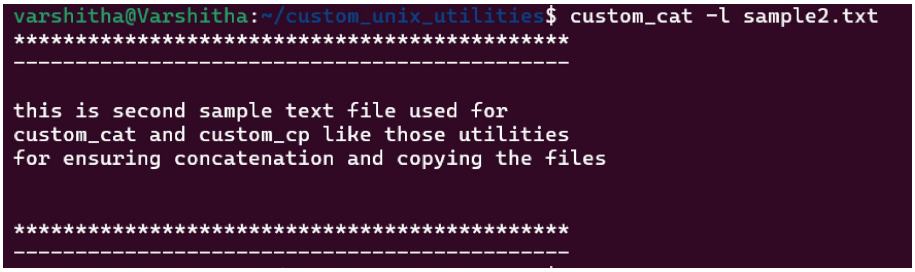

Figure 9: custom_cat -l demonstration

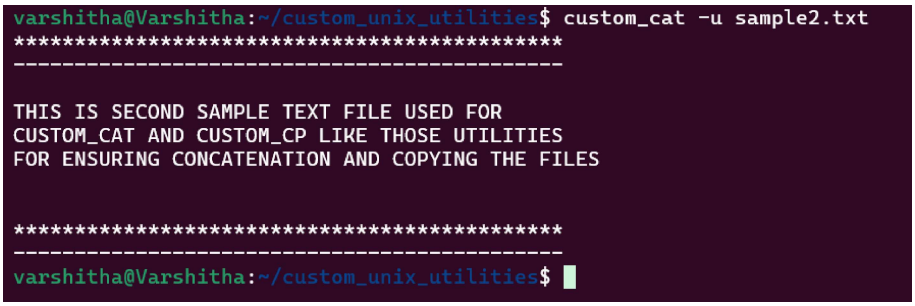

Figure 10: custom_cat -u demonstration

4.5 custom_grep

Description: Searches for a pattern in files, similar to the `grep` command.

Functionalities:

- Searches for lines matching a given pattern.
- Supports case-insensitive search (`-i`).
- Displays lines that do not match the pattern using the `-v` flag.

Key Concepts:

- **Pattern Matching:** Implemented using `strstr`.
- **Case Insensitivity:** Achieved by converting strings to lowercase using `tolower`.

Usage:

```
custom_grep [-i|-v] pattern file1 file2 ...
# Examples:
custom_grep "pattern" file1.txt     # Search for a pattern in file1.txt
    .
custom_grep -i "pattern" file1.txt  # Perform case-insensitive search.
custom_grep -v "pattern" file1.txt  # Perform case-insensitive search
    and displays all the lines which are not matched.
```

4.6 Demonstration

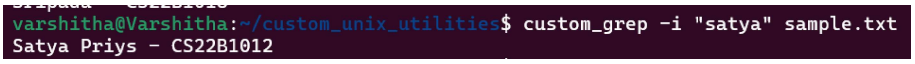

Figure 11: custom_grep -i demonstration

Figure 12: custom_grep -v demonstration

4.7 custom_wc

Description: Counts words, lines, and characters in a file, similar to the `wc` command.

Functionalities:

- Counts and displays the number of lines, words, and characters in a file.
- Processes multiple files with individual and total counts.

Usage:

```
custom_wc file1 file2 ...
# Examples:
custom_wc file1.txt                 # Count lines, words, and
    characters in file1.txt.
custom_wc file1.txt file2.txt       # Count for multiple files.
```

4.8 Demonstration

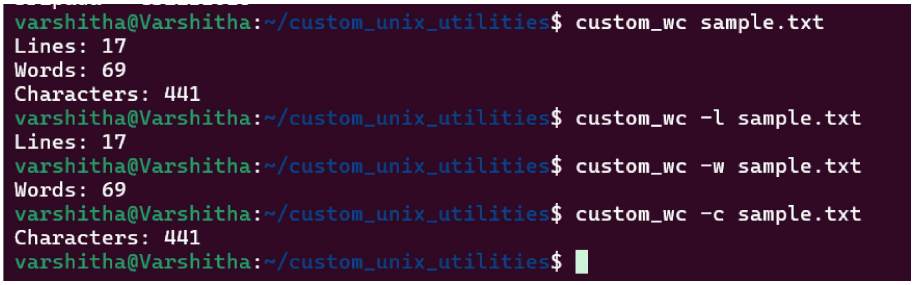

Figure 13: custom_wc demonstration

4.9 custom_cp

Description: Copies files or directories, similar to the `cp` command.

Functionalities:

- Copies a single file from source to destination.
- Recursively copies directories using the `-r` flag.

Usage:

```
custom_cp [-r] source destination
# Examples:
custom_cp file1.txt file2.txt        # Copy file1.txt to file2.txt.
custom_cp -r dir1 dir2               # Copy dir1 to dir2 recursively.
```

4.10 Demonstration

Figure 14: Performing ls before executing custom_mv utility

Here we performed custon_mv. We can see that after ls it has listed the newly created file file_cpy.txt

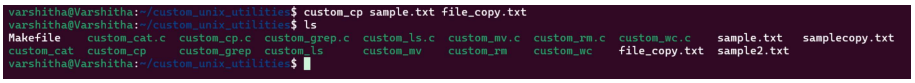

Figure 15: Performing ls after executing custom_mv utility

Here we can see that after performing custom_mv it has displayed the original file content in the newly copied file(file_copy.txt) using custom_cat

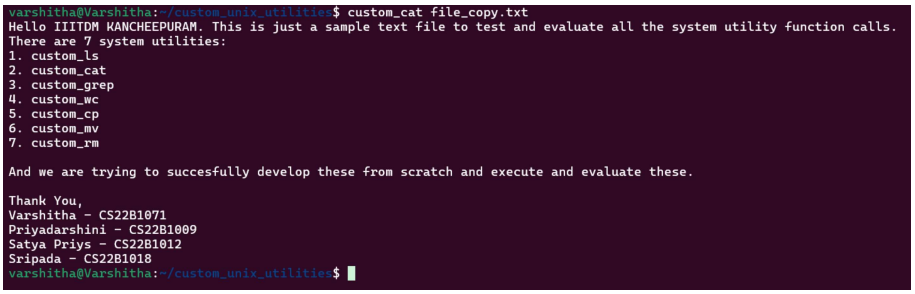

Figure 16: Copied file(file_copy.txt) content

The same procedure is performed with directories. Below are the images for demonstration for directories.

Figure 17: ls before copying

Here we can see that after performing custom_cp the new directory is listed.

Figure 18: ls after copying

Here we can see that the same original file is there inside newly created file and also the content in this is same as original file content.

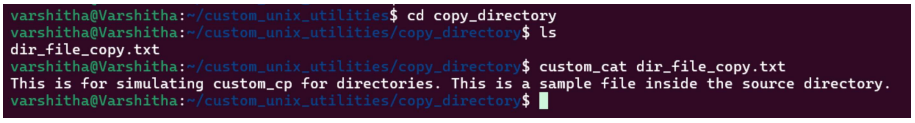

Figure 19: After copying the directory

4.11 custom_mv

Description: Moves or renames files and directories, similar to the `mv` command.

Functionalities:

- Renames a file or directory.
- Moves a file or directory to a new location.

Usage:

```
custom_mv source destination
# Examples:
custom_mv old.txt new.txt           # Rename old.txt to new.txt.
custom_mv dir1 /home/user/dir2      # Move dir1 to a new location.
```

4.12 Demonstration

Figure 20: ls before performing moving

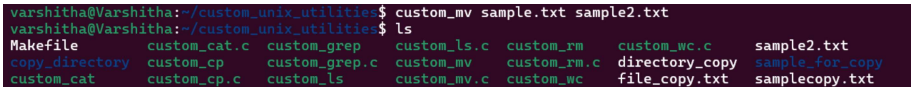

Figure 21: ls after performing moving

Here we can see that the sample.txt content is moved into sample2.txt.

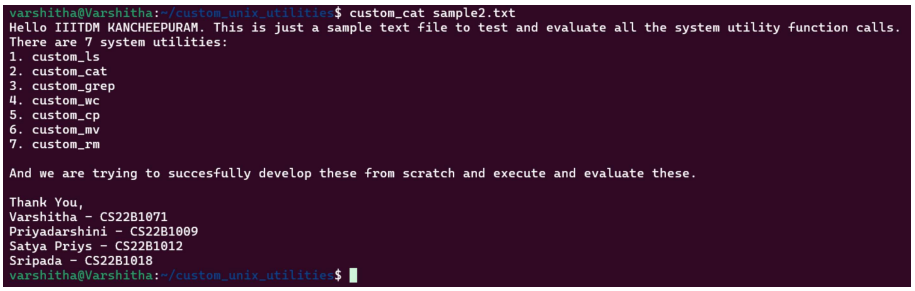

Figure 22: Displaying content of sample2.txt

4.13 custom_rm

Description: Deletes files or directories, similar to the `rm` command.

Functionalities:

- Deletes files.
- Recursively deletes directories using the `-r` flag.

Usage:

```
custom_rm [-r] file_or_directory
# Examples:
custom_rm file1.txt                    # Delete file1.txt.
custom_rm -r dir1                      # Recursively delete dir1.
```

4.14 Demonstration

Here we can see that after performing custom_rm file_copy.txt that file is not listed which represents it has removed

Figure 23: ls after performing custom_rm

5 Custom Shell Program

A custom shell program was developed to execute these utilities interactively. Users can type commands to invoke the utilities, which are processed and executed by the shell.

6 Conclusion

This project provided hands-on experience with system programming concepts and file operations. By re-implementing standard UNIX commands, we gained a deeper understanding of their functionality and inner workings.