Simplified UNIX-Like Utilities

Project Documentation

November 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
  ```

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
  ```

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/username/custom_unix_utilities
```

Explanation:

- The `export` command temporarily appends `/home/username/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
  ```

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/username/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:

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/username/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

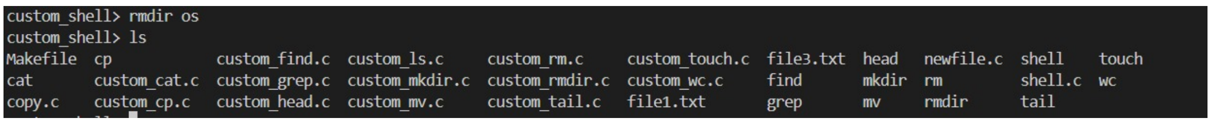
4.1 custom_ls

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 `-R` flag.
- Displays hidden files using the `-a` flag.

Key Concepts:

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



```
custom_shell> rmdir os
custom_shell> ls
Makefile  cp           custom_find.c  custom_ls.c     custom_rm.c     custom_touch.c  file3.txt  head   newfile.c  shell    touch
cat       custom_cat.c custom_grep.c  custom_mkdir.c  custom_rmdir.c  custom_wc.c     find       mkdir  rm         shell.c  wc
copy.c    custom_cp.c  custom_head.c  custom_mv.c     custom_tail.c   file1.txt       grep       mv     rmdir      tail
```

Figure 1: customls

4.2 Demonstration

The following figure illustrates how the

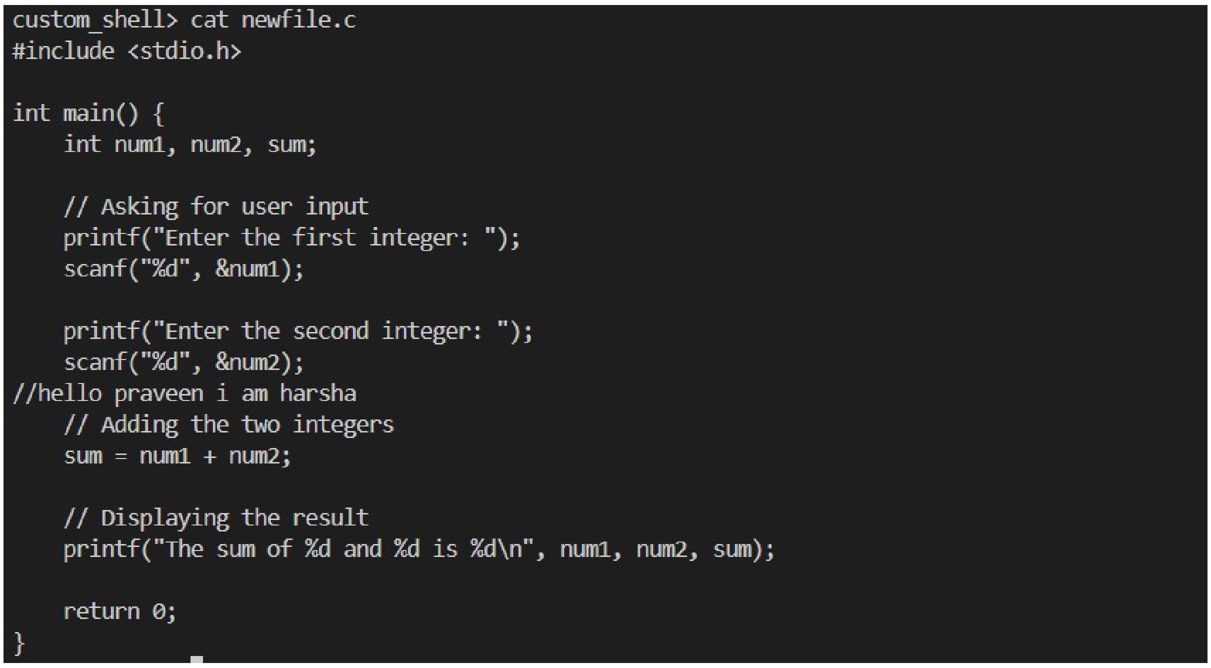
4.3 custom_cat

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`.



```
custom_shell> cat newfile.c
#include <stdio.h>

int main() {
    int num1, num2, sum;

    // Asking for user input
    printf("Enter the first integer: ");
    scanf("%d", &num1);

    printf("Enter the second integer: ");
    scanf("%d", &num2);
//hello praveen i am harsha
    // Adding the two integers
    sum = num1 + num2;

    // Displaying the result
    printf("The sum of %d and %d is %d\n", num1, num2, sum);

    return 0;
}
```

Figure 2: customcat

4.4 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`.

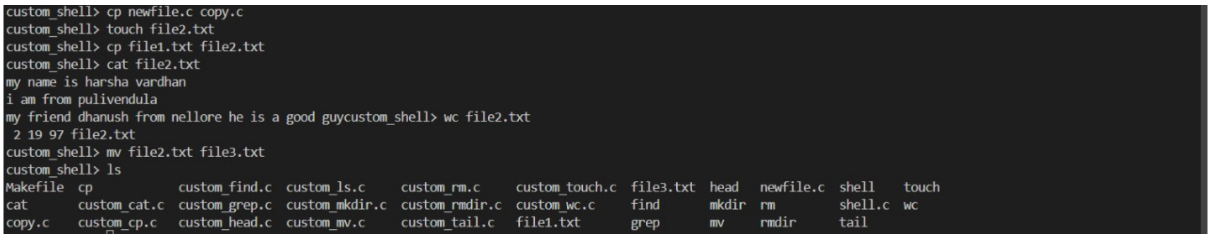
Usage:

```
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.
```

4.5 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.



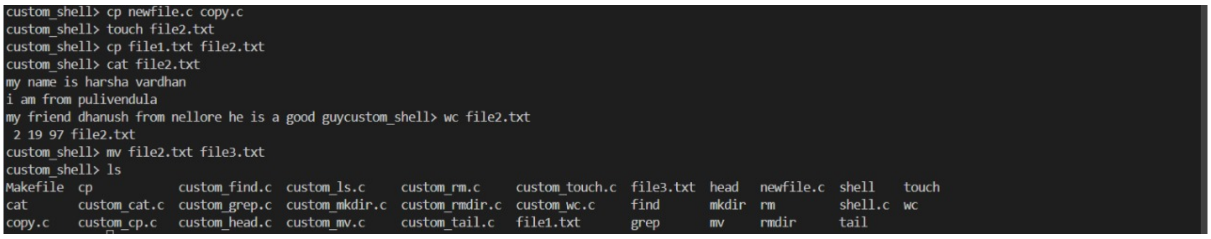
```
custom_shell> cp newfile.c copy.c
custom_shell> touch file2.txt
custom_shell> cp file1.txt file2.txt
custom_shell> cat file2.txt
my name is harsha vardhan
i am from pulivendula
my friend dhanush from nellore he is a good guycustom_shell> wc file2.txt
 2 19 97 file2.txt
custom_shell> mv file2.txt file3.txt
custom_shell> ls
Makefile  cp          custom_find.c  custom_ls.c     custom_rm.c     custom_touch.c  file3.txt  head   newfile.c  shell    touch
cat       custom_cat.c  custom_grep.c  custom_mkdir.c  custom_rmdir.c  custom_wc.c     find       mkdir  rm         shell.c  wc
copy.c    custom_cp.c   custom_head.c  custom_mv.c     custom_tail.c   file1.txt       grep       mv     rmdir      tail
```

Figure 3: custom/wc

4.6 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.



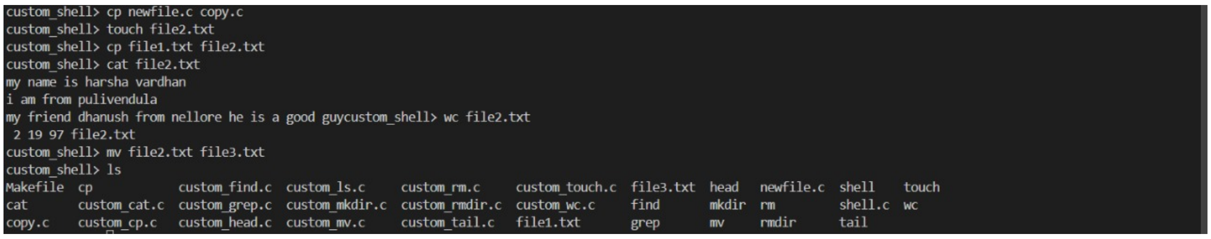
```
custom_shell> cp newfile.c copy.c
custom_shell> touch file2.txt
custom_shell> cp file1.txt file2.txt
custom_shell> cat file2.txt
my name is harsha vardhan
i am from pulivendula
my friend dhanush from nellore he is a good guycustom_shell> wc file2.txt
 2 19 97 file2.txt
custom_shell> mv file2.txt file3.txt
custom_shell> ls
Makefile  cp          custom_find.c  custom_ls.c     custom_rm.c     custom_touch.c  file3.txt  head   newfile.c  shell    touch
cat       custom_cat.c  custom_grep.c  custom_mkdir.c  custom_rmdir.c  custom_wc.c     find       mkdir  rm         shell.c  wc
copy.c    custom_cp.c   custom_head.c  custom_mv.c     custom_tail.c   file1.txt       grep       mv     rmdir      tail
```

Figure 4: custom/cp

4.7 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.



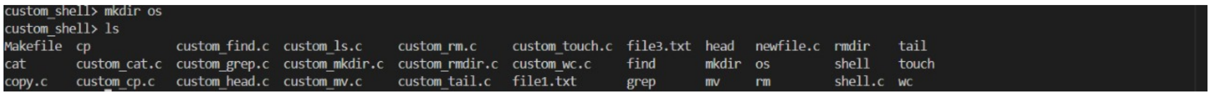
```
custom_shell> cp newfile.c copy.c
custom_shell> touch file2.txt
custom_shell> cp file1.txt file2.txt
custom_shell> cat file2.txt
my name is harsha vardhan
i am from pulivendula
my friend dhanush from nellore he is a good guycustom_shell> wc file2.txt
 2 19 97 file2.txt
custom_shell> mv file2.txt file3.txt
custom_shell> ls
Makefile  cp          custom_find.c  custom_ls.c     custom_rm.c     custom_touch.c  file3.txt  head   newfile.c  shell    touch
cat       custom_cat.c  custom_grep.c  custom_mkdir.c  custom_rmdir.c  custom_wc.c     find       mkdir  rm         shell.c  wc
copy.c    custom_cp.c   custom_head.c  custom_mv.c     custom_tail.c   file1.txt       grep       mv     rmdir      tail
```

Figure 5: custom/mv

4.8 custom_rm

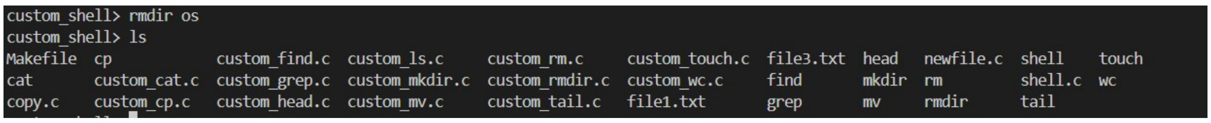
Description: Deletes files or directories, similar to the `rm` command.
Functionalities:

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



```
custom_shell> mkdir os
custom_shell> ls
Makefile  cp          custom_find.c  custom_ls.c     custom_rm.c     custom_touch.c  file3.txt  head   newfile.c  rmdir    tail
cat       custom_cat.c custom_grep.c  custom_mkdir.c  custom_rmdir.c  custom_wc.c     find       mkdir  os         shell    touch
copy.c    custom_cp.c  custom_head.c  custom_mv.c     custom_tail.c   file1.txt       grep       mv     rm         shell.c  wc
```

Figure 6: custom/mkdir



```
custom_shell> rmdir os
custom_shell> ls
Makefile  cp          custom_find.c  custom_ls.c     custom_rm.c     custom_touch.c  file3.txt  head   newfile.c  shell    touch
cat       custom_cat.c custom_grep.c  custom_mkdir.c  custom_rmdir.c  custom_wc.c     find       mkdir  rm         shell.c  wc
copy.c    custom_cp.c  custom_head.c  custom_mv.c     custom_tail.c   file1.txt       grep       mv     rmdir      tail
```

Figure 7: custom/rmdir

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.