
GHULAM ISHAQ KHAN INSTITUTE OF ENGINEERING SCIENCES AND TECHNOLOGY TOPI,SWABI

# Report for the Custom Terminal Project in C Language

**SUBMITTED TO**
**MAAM MAHRUKH**

**SUBMITTED BY**
MUSKAN AHMAD
2022468
MAHRUKH FATIMA
2022273
KANEEZ E ZAHRA
2022245

**DATE: 10-DEC-2024**

# Introduction

THE MYSH PROJECT INVOLVES THE DEVELOPMENT OF A CUSTOM TERMINAL SHELL WRITTEN IN C. THE GOAL IS TO CREATE A LIGHTWEIGHT AND SIMPLE TERMINAL THAT SIMULATES THE BEHAVIOR OF COMMON UNIX/LINUX SHELLS. THE TERMINAL SUPPORTS EXECUTING EXTERNAL COMMANDS, BUILT-IN COMMANDS LIKE CD AND EXIT, INPUT/OUTPUT REDIRECTION, AND PIPING. IT IS A FOUNDATIONAL TOOL FOR LEARNING ABOUT PROCESS CONTROL, SYSTEM CALLS, AND BASIC SHELL FUNCTIONALITIES IN OPERATING SYSTEMS.

# Features

THE CUSTOM TERMINAL, MYSH, IMPLEMENTS SEVERAL FEATURES TYPICAL OF A LINUX SHELL. THE FUNCTIONALITIES INCLUDE:

BUILT-IN COMMANDS:

- EXIT: EXITS THE SHELL PROGRAM, TERMINATING THE USER SESSION.
- CD <DIRECTORY>: CHANGES THE CURRENT WORKING DIRECTORY TO THE SPECIFIED DIRECTORY. THIS COMMAND MIMICS THE FUNCTIONALITY OF THE CD COMMAND IN BASH.

EXTERNAL COMMAND EXECUTION

- THE SHELL SUPPORTS RUNNING ANY EXTERNAL COMMAND AVAILABLE IN THE SYSTEM'S PATH, SUCH AS LS, PWD, CAT, AND MORE. THESE COMMANDS ARE EXECUTED IN CHILD PROCESSES CREATED BY FORK(), AND THEIR OUTPUT IS DISPLAYED IN THE TERMINAL.

INPUT AND OUTPUT REDIRECTION

- OUTPUT REDIRECTION (>): THE SHELL ALLOWS THE OUTPUT OF A COMMAND TO BE REDIRECTED INTO A FILE, OVERWRITING THE EXISTING FILE IF IT EXISTS.
- INPUT REDIRECTION (<): THE SHELL ALSO ALLOWS READING FROM A FILE TO PROVIDE INPUT TO COMMANDS.

PIPING

- THE SHELL SUPPORTS PIPING, WHERE THE OUTPUT OF ONE COMMAND IS USED AS THE INPUT FOR THE NEXT. THIS IS ACHIEVED BY CONNECTING COMMANDS THROUGH PIPES (|).

PROCESS MANAGEMENT

- THE TERMINAL ALLOWS COMMANDS TO BE EXECUTED IN THE BACKGROUND USING &. THIS PREVENTS THE TERMINAL FROM WAITING FOR THE PROCESS TO FINISH BEFORE ACCEPTING NEW COMMANDS.

# DESIGN AND IMPLEMENTATION

## Main Loop

THE MAIN LOOP READS USER INPUT, PARSES THE COMMAND, AND DETERMINES THE TYPE OF COMMAND (WHETHER IT'S BUILT-IN, EXTERNAL, INVOLVES REDIRECTION, OR USES PIPING).

```c
int main() {
    char input[1024];

    while (1) {
        // Display prompt
        printf("mysh> ");
        fflush(stdout);

        // Read user input
        if (fgets(input, sizeof(input), stdin) == NULL) {
            printf("\nExiting...\n");
            break;
        }

        // Remove trailing newline
        input[strcspn(input, "\n")] = '\0';

        // Check for empty input
        if (strlen(input) == 0) continue;

        // Handle piping
        if (strchr(input, '|')) {
            executeWithPiping(input);
        } else {
            // Parse and execute command
            executeCommand(input);
        }
    }

    return 0;
}
```

# Command Execution

WHEN A COMMAND IS ENTERED, THE EXECUTECOMMAND FUNCTION IS RESPONSIBLE FOR PARSING AND EXECUTING IT.

```c
void executeCommand(char *input) {
    char *args[100];
    pid_t pid;

    // Parse the command into arguments
    parseCommand(input, args);

    // Handle built-in commands like "cd" and "exit"
    if (handleBuiltInCommands(args)) return;

    // Handle redirection
    if (strchr(input, '>')) {
        executeWithRedirection(args);
        return;
    }

    // Create a child process
    pid = fork();

    if (pid < 0) {
        perror("Error: Fork failed");
    } else if (pid == 0) {
        // Child process
        if (execvp(args[0], args) == -1) {
            perror("Error: Command not found");
        }
        exit(EXIT_FAILURE);
    } else {
        // Parent process
        wait(NULL);
    }
}
```

# Redirection Handling

REDIRECTION (>, <) IS IMPLEMENTED BY MODIFYING THE FILE DESCRIPTORS USING DUP2().

```c
void executeWithRedirection(char **args) {
    int i = 0;
    while (args[i] != NULL) {
        if (strcmp(args[i], ">") == 0) {
            args[i] = NULL; // Split the command at '>'
            int fd = open(args[i + 1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
            if (fd < 0) {
                perror("Error opening file");
                return;
            }
            dup2(fd, STDOUT_FILENO); // Redirect stdout
            close(fd);
            break;
        }
        i++;
    }

    if (execvp(args[0], args) == -1) {
        perror("Error: Command not found");
    }
    exit(EXIT_FAILURE);
}
```

# Piping

```c
void executeWithPiping(char *input) {
    char *commands[2];
    char *args1[100], *args2[100];
    int fd[2];

    commands[0] = strtok(input, "|");
    commands[1] = strtok(NULL, "|");

    pipe(fd);
    if (fork() == 0) {
        // First command
        dup2(fd[1], STDOUT_FILENO);
        close(fd[0]);
        close(fd[1]);
        parseCommand(commands[0], args1);
        if (execvp(args1[0], args1) == -1) {
            perror("Error: Command not found");
        }
        exit(EXIT_FAILURE);
    }
    if (fork() == 0) {
        // Second command
        dup2(fd[0], STDIN_FILENO);
        close(fd[0]);
        close(fd[1]);
        parseCommand(commands[1], args2);
        if (execvp(args2[0], args2) == -1) {
            perror("Error: Command not found");
        }
        exit(EXIT_FAILURE);
    }
    close(fd[0]);
    close(fd[1]);
    wait(NULL);
```

# Background Execution

**Background execution is implemented by checking for the & symbol at the end of a command**
**If a command ends with &, the shell executes it in the background without waiting for the process to finish.**

# EXAMPLES OF COMMANDS

```
mysh> ls
file  sample_data  terminal  terminal.c
mysh> ls -l
total 36
drwxr-xr-x 2 root root  4096 Dec  9 14:13 file
drwxr-xr-x 1 root root  4096 Dec  5 14:24 sample_data
-rwxr-xr-x 1 root root 17208 Dec  9 14:21 terminal
-rw-r--r-- 1 root root  5298 Dec  9 14:19 terminal.c
mysh> ls -a
.  ..  .config  file  .ipynb_checkpoints  sample_data  terminal  terminal.c
mysh> pwd
/content
mysh> touch newfile.txt
mysh> cat newfile.txt
mysh> echo "Hello, world!"
"Hello, world!"
mysh> nano newfile.txt
Error: Command not found: No such file or directory
mysh> echo "Hello, world!" > output.txt ls > directory_listing.txt
"Hello, world!" > output.txt ls > directory_listing.txt
mysh> cat < input.txt
```

# Applications

**Highlight practical applications of building a shell:**

- **Learning Purpose: Understand process control, system calls, and inter-process communication.**
- **Foundation for Advanced Shells: Build more complex tools like task automation or custom scripting environments.**

# Conclusion

**The mysh project is a basic shell implementation that demonstrates core features of a terminal, including command execution, built-in commands, redirection, piping, and background processes. While it lacks some of the advanced features of a full Linux shell, it provides a solid foundation for understanding how shells operate at a system level. Further improvements can be made to add more functionality, robustness, and user-friendly features.**