Report on Command-Line Interface (CLI) Shell Implementation in C

Your Name
Your Institution
Your Email

***Abstract*—This C program implements a simple command-line interface (CLI) shell for Windows, resembling popular Unix/Linux shells like Bash. The shell handles user input, manages command history, changes directories, lists files, and performs basic file system operations. The primary aim is to allow users to execute fundamental commands (cd, ls, pwd, echo, hist) and interact with the Windows operating system through a terminal interface.**

***Index Terms*—CLI, Shell, C, Windows, Command History, Directory Navigation, File Listing**

I. Introduction

This document outlines the implementation of a command-line interface (CLI) shell in C for the Windows platform. The shell emulates the functionality of popular Unix/Linux shells like Bash, providing users with a terminal-based interface to interact with the operating system. Key features include command history management, directory navigation, file listing, and execution of basic commands.

II. Program Structure and Components

A. Libraries and Macros

The program utilizes several standard C libraries:

- `stdio.h`: For standard input/output functions.
- `stdlib.h`: Provides standard library functions like `getenv()` and `strtok()`.
- `string.h`: Offers functions for string manipulation (e.g., `strncpy()`, `strcmp()`).
- `windows.h`: Includes Windows-specific system calls (e.g., `GetComputerName()`, `FindFirstFile()`).
- `direct.h`: Contains directory-related functions like `_getcwd()` and `_chdir()`.
- `limits.h`: Defines `PATH_MAX` (the maximum allowable path length).

Macros used in the program include:

- `MAX_INPUT`: Defines the maximum size for input (1024 characters).
- `MAX_ARGS`: Sets the maximum number of command arguments (64).
- `MAX_HISTORY`: Specifies the size of the command history buffer (10).

B. Command History Management

The shell maintains a history of the last 10 commands executed by the user.

1) Global Variables:

- `history[MAX_HISTORY][MAX_INPUT]`: Stores the history of the last 10 commands.
- `history_count`: Tracks the current number of commands in the history.

2) Functions:

- `add_to_history(const char *command)`: Adds the latest command to the history. If the history buffer is full, the oldest command is removed, and the new command is added.
- `show_history()`: Displays the command history to the user.

C. Directory Navigation

The program allows users to change and print the current working directory.

1) Functions:

- `change_directory(char *path, char *prev_dir)`: Changes the current working directory. If no path is provided or the user specifies " ", it defaults to the user's home directory. If the user enters "-", the program switches to the previous directory.
- `print_working_directory()`: Prints the current working directory using `_getcwd()`.

D. File Listing

The program provides a function to list the files in the current directory:

1) Functions:

- `list_files()`: Lists the files in the current directory using Windows API functions. It utilizes `FindFirstFile()` and `FindNextFile()` to traverse files and directories in the current location.

E. Command Parsing and Execution

The shell reads user input, tokenizes the command, and executes appropriate actions.

1) Input Parsing:

- The shell continuously waits for user input via `fgets()`.
- The input is split into tokens using `strtok()`, and each token is stored in the `args` array.
- Commands are matched using `strcmp()` and executed based on a set of predefined commands.

2) Supported Commands:

- `cd`: Changes the current directory. Handles special cases like `cd -` (switch to the previous directory) and `cd` (switch to the home directory).
- `ls`: Lists files in the current directory.
- `pwd`: Prints the current working directory.
- `echo`: Prints the provided arguments to the terminal.
- `hist`: Displays the command history.
- `exit`: Exits the shell program.

F. Command Execution Workflow

1) Prompt Display: The shell prints a prompt in the format `[username@hostname cwd$]`, where:

- `username`: Retrieved from the environment variable `USERNAME`.
- `hostname`: Obtained using the Windows function `GetComputerName()`.
- `cwd`: The current working directory, fetched using `_getcwd()`.

2) Input Handling:

- The shell reads user input and processes it with `fgets()`.
- If the command is not empty, it is added to the history using `add_to_history()`.

3) Command Execution:

- The shell tokenizes the input into commands and arguments.
- It checks for each supported command (cd, ls, pwd, echo, etc.) and invokes the corresponding function.
- If an unsupported command is entered, the shell prints an error message.

4) Looping: The program continuously loops until the user types `exit`.

III. Error Handling

The shell performs basic error handling:

- If `getcwd()` or `chdir()` fails, it prints an appropriate error message.
- If too many arguments are passed to the `cd` command, the program prints an error and ignores the command.
- The `FindFirstFile()` function handles cases where the directory cannot be read.

IV. Key Functions

- `change_directory(char *path, char *prev_dir)`: Handles changing directories and supports navigating to the home directory ( ) and switching to the previous directory (-).
- `list_files()`: Uses Windows API functions to list files and directories in the current location.
- `echo_command(char *input)`: This function prints the users input to the terminal, implementing an echo command.

V. Conclusion

This program illustrates the implementation of a simple command-line shell in C for a Windows system. It incorporates essential features such as command history, directory navigation, file listing, and the ability to execute basic commands. The use of Windows API functions facilitates interaction with the file system, while standard C functions handle input/output and string manipulation.

The shell could be further enhanced by adding support for more commands and features like piping, file redirection, or job control to increase its functionality and user experience.