OS Assignment-2 of Group No. 44 (Section-A) by Prof. Vivek Kumar

Simple-Shell

Introduction

This Simple Shell program is designed to mimic the functionality of a basic command-line shell, similar to the Unix/Linux shell. It provides a user-friendly interface for executing commands, tracks command execution history, and can display detailed information about past commands.

Code Structure

The code is divided into several sections, each serving a specific purpose.

Header Includes

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
```

These header files are included to provide access to various C library functions and system calls used in the program.

Constants

```
#define MAX_INPUT_LENGTH 1024
#define MAX_HISTORY_SIZE 50
```

These constants define the maximum length of user input and the maximum number of commands to be stored in the history.

Struct: CommandExecution

```
struct CommandExecution {
    char cmd[MAX_INPUT_LENGTH];
    pid_t pid;
    time_t start_time;
    time_t end_time;
    double duration;
};
```

This structure is used to store details about each command execution, including the command itself (`cmd`), the process ID (`pid`), the start time (`start_time`), the end time (`end_time`), and the execution duration (`duration`).

Function: launch

```
void launch(char *cmd, struct CommandExecution *history, int *history_count);
```

- The `launch` function is responsible for executing user-entered commands.
- It records the start time, forks a child process, and executes the command in the child.
- The parent process waits for the child to finish unless the command ends with `&`, indicating a background process.
- After execution, it records the end time, calculates the execution duration, and updates the command history.

Function: main

```
int main()
```

The `main` function is the entry point of the program and contains the main logic for the Simple Shell.

Initialization

- It initializes variables, including an input buffer (`input`), an array to store command execution history (`history`), and a counter for the number of history entries (`history_count`).
- The program starts by displaying a simple prompt: "SimpleShell> ".

Command Execution Loop

- The main loop continues until the user types "exit" or "quit" (terminated by Ctrl+D or EOF).
- User-entered input is trimmed of trailing newline characters.

- If the input is "exit" or "quit," the program displays the execution details of all previously executed commands and exits the shell.
- If the input is "history," it displays the command history.
- Otherwise, it stores the command in the history (with a limited size of `MAX_HISTORY_SIZE`), launches it, and updates the history count.
- After processing each command, the program displays the shell prompt for the next input.

Example Usage

1. For running the program you have to generate executables of simple-shell.c by “gcc -o simple-shell simple-shell.c” command.

2. Then you can run the program using “./simple-shell” command.

3. That should display “SimpleShell>” , now you are ready to use a simple shell.

4. It supports commands like:
   - ls
   - ls -R
   - ls -l
   - echo
   - ls /home
   - wc -l fib.c
   - wc -c fib.c
   - grep printf helloworld.c
   - ./fib 40
   - ./helloworld
   - sort fib.c
   - uniq file.txt
   - cat fib.c | wc -l
   - cat helloworld.c | grep printf | wc -l
   - pwd
   - mkdir
   - rmdir
   - rm file.txt
   - cd
   - touch file.txt
   - chmod +x file
   - df -h (shows information about total size, used space, available space, and usage percentage, all displayed in human-readable sizes.)
   - head -n 5 file.txt //shows first few lines
   - tail -n 5 file.txt //shows last few lines

3. And Some Bonus Commands:

- “ls &” for running background process
- “./myscript.sh” for running shell script

4. I have also provided some additional files such as “file.txt” , “fib.c” , “helloworld.c” and “myscript.sh” along with their executables which are just for demonstration purposes and can be changed according to user requirements. You can generate their executable files with “gcc -o fib fib.c” and “gcc -o helloworld helloworld.c”.

5. Code also supports history by “history” command.

6. On terminating the SimpleShell it should display additional details on the execution of each command, e.g., process pid, time at which the command was executed, total duration the command took for execution, etc.

Contributors

1.) **Athiyo Chakma:** Developed the core functionality for command execution and process management.
2.) **Yash Goyal:** Implemented command history tracking and provided documentation for the program.

Conclusion

This Simple Shell program provides a basic command-line interface for executing commands and managing a history of executed commands. It can be used as a starting point for educational purposes and extended to include more advanced features like custom command parsing or shell scripting capabilities.