Operating Systems Coursework - C Shell

Galen Han

Contents

Features

- Load `$HOME` and `$PATH` from `profile` file
- `$HOME` and `$PATH` variable assignment
- Execute any commands located in `$PATH`

Implementation

definition.h

Holds the struct definition of the `Shell` struct.

```c
typedef struct Shell {
    char cwd[4097]; //Max path length in Linux is 4096

    char **path; //Array of directories
    char *home;
} Shell;
```

Stores the state of the shell, i.e. the current working directory, current `$PATH` and `$HOME`.

main.c

The main function initialises the shell by calling ***load_profile()*** which reads the `profile` file and initialises the `$HOME` and `$PATH` variables.

It then calls the `command_loop` function which calls subroutines to:

- Print the current working directory to the output
- ***read_line()*** - Wait for input from the user and read it

- ***parse_args()*** - Split the input into args
- Exit the terminal if `exit` is entered
- ***execute_cmd()*** - Execute any other commands with their respective arguments

executeCmd.c

Inside the file ***executeCmd.c*** we have the function `execute_cmd()` which given the state of the shell executes a command.

First it checks if the command is a builtin. These are:

- `cd` - Changes the current working directory of the shell.
- `$VAR=` - Environment variable assignments

The code to handle builtins is defined in ***builtin.c***.

If the command is not a builtin:

- Spawn a new child process using `fork()`
- If we're in the child process:
  - Look for the program in the current `$PATH` within `shell->path` with the function `find_program()`.
  - If the return value of `find_program()` is `NULL`, then the program could not be found within the `$PATH` and an error is returned to the shell.
  - Else, we use the path of the found executable and call `execv()` to run the program in the child process. Any output from the program is printed to the terminal.
- If we're in the parent process:
  - Wait until the child process terminates

builtin.c

cd

The cd builtin function takes the state of the shell and the path to change to as a string as its parameters.

We try to change the directory using `chdir(path)`. It return `-1` if it fails.

If it does not fail we get the new current working directory using `getcwd()` and update the state of the shell `shell->cwd` to the new path.

$VAR=

Within `set_shell_variable()` we simply call `set_variable()` from loadProfile.c

inputHandler.c

This file contains all the logic for reading input to the shell, allocating memory and parsing the input correctly.

read_line()

This function reads a line from the shell input and returns a pointer to a string.

We use a buffer that gets dynamically reallocated when it needs more memory.

parse_args(char *line)

This splits the input line into an array of arguments when there is a space or other delimiting token.

We use `strtok()` to do most of the work.

loadProfile.c

This file includes logic for reading and parsing the `profile` file.

load_profile()

Loads the `profile` file using `fopen()` and reads it line by line using `getline()`.

We parse and set the `$HOME` and `$PATH` variables and store it in the shell state.

If no `$HOME` or `$PATH` is set, an error is thrown and the shell exits.

set_variable()

Sets the shell state for the `$PATH` and `$HOME` variable assignments.