CS-392 Systems Programming — Spring 2023

Assignment 5 · Minishell

Lecturer: Shudong Hao — *Date: See Canvas*

1 Objective

For this assignment your goal is to write a basic shell program in C with a few built-in commands. This shell must implement color printing, `cd` and `exit` commands as built-ins, signal handling, and fork/exec for all other commands.

This program will involve making a shell. The main loop of this function should print the current working directory in blue, then wait for user input. When the user input has been entered, the program should act accordingly. The program will be called from the command line and will take no arguments.

2 Main Task

2.1 Prompt/Color Printing

The prompt always displays the current working directory in square brackets followed immediately by a `>` and a space. The directory should be printed in **blue**. Place these color definitions beneath the includes of your `minishell.c` file:

```
#define BLUE       "\x1b[34;1m"
#define DEFAULT    "\x1b[0m"
```

These strings will allow you to print to the terminal in blue or the default color for text, based on the terminal's properties. For example, to print just the string `"Hello, world!"` in blue, you would write:

```
printf("%sHello, world!\n", BLUE);
```

Adding the escape sequence to your string will change the color of anything printed after it, so if you want to return to printing in the default terminal color, make sure you switch it back in the `printf()` function call:

```
printf("%sHello, world! in blue%s\n", BLUE, DEFAULT);
printf("I'm back!\n");
```

2.2 Supported Commands

You will have to manually implement the following commands.

2.2.1 cd

If `cd` is called with no arguments or the one argument ~, it should change working directory to the user's home directory. Do NOT hard code any specific folder as the home directory. Instead work with the following:

- `getuid()`;
- `getpwuid()`;
- `chdir()`.

If `cd` is called with one argument that isn't ~, it should attempt to open the directory specified in that argument.

2.2.2 exit

The `exit` command should cause the shell to terminate and return `EXIT_SUCCESS`. Using `exit` is the only way to stop the shell's execution normally.

2.2.3 Other Commands

All other commands will be executed using `exec()`. When an unknown command is entered, the program forks. The child program will `exec()` the given command and the parent process will wait for the child process to finish before returning to the prompt.

2.2.4 Error Handling

Errors for system/function calls should be handled with printing error messages to `stderr`. At a minimum, you will need to incorporate the following error messages into your shell. The last `%s` in each line below is a format specifier for `strerror(errno)`.

```
"Error: Cannot get passwd entry. %s.\n"
"Error: Cannot change directory to %s. %s.\n"
"Error: Too many arguments to cd.\n"
"Error: Failed to read from stdin. %s.\n"
```

```
"Error: fork() failed. %s.\n"
"Error: exec() failed. %s.\n"
"Error: wait() failed. %s.\n"
"Error: malloc() failed. %s.\n" // If you use malloc()
"Error: Cannot get current working directory. %s.\n"
"Error: Cannot register signal handler. %s.\n"
```

If you use other system/function calls, follow a similar format for the corresponding error messages.

3 Signals

Your minishell needs to capture the `SIGINT` signal. Upon doing so, it should return the user to the prompt. Interrupt signals generated in the terminal are delivered to the active process group, which includes both parent and child processes. The child will receive the `SIGINT` and deal with it accordingly.

One suggestion would be to use a single **volatile** `sig_atomic_t` variable called `interrupted` that is set to true inside the signal handler. Then, inside the program's main loop that displays the prompt, read the input, and execute the command, don't do anything if that iteration of the loop was interrupted by the signal. If read fails, you need to make sure it wasn't simply interrupted before erroring out of the minishell. Finally, set `interrupted` back to false before the next iteration of the main loop.

To declare a volatile integer, simply use:

```
volatile int a;
```

4 Sample Run

You don't need to worry about empty lines or spaces.

Sample Run

```
$ ./minishell
[/home/user/minishell]> echo HI
HI
[/home/user/minishell]> cd ..
[/home/user/]> cd minishell
[/home/user/minishell]> cd /tmp
[/tmp]> cd ~
[/home/user/]> cd minishell
[/home/user/minishell]> ls
makefile
minishell
minishell.c
[/home/user/minishell]> pwd
/home/user/minishell
[/home/user/minishell]> cd
[/home/user/]> pwd
/home/user
[/home/user/]> nocommand
Error: exec() failed. No such file or directory.
[/home/user/]> cd minishell
[/home/user/minishell]> ^C
[/home/user/minishell]> sleep 10
^C
[/home/user/minishell]> exit
$
```

5 Requirement

Note that requirements listed here are only part of the rubrics. Meeting the requirements doesn't mean you'll get full credits. If you have something not sure or clear about, ask the TA or the instructor.

- Utilize `fprintf()` with `stderr` and `strerror` for error messages. Do not use `printf()`;
- Use `getuid()`, `getpwuid()`, and/or `chdir()`. Do not hard code specific folder as home directory;
- Do not use `signal()` function; use `sigaction()` instead;

- Make sure to do error checks for all C function calls in your code;
- Please write your name and your pledge to Steven's honor system at the beginning of your code;
- Your code should compile and have no memory leak;
- Your code should have appropriate and useful comments;
- If you implemented extra credit items, or there's some library we need to link while compiling your code, point them out when you submit your homework on Canvas.

6 What To Submit

Submit a single `.c` file through Canvas.

A Extra Credits

Upon successful completion of the main task, you're encouraged to implement other commands. Instead of using `exec()` to invoke existing binaries, you can use all the structures and functions we learned to implement them.

A.1 5 Point Items

- Colorized `ls`: for this command, you'd list all the files given a directory. If there's no argument following `ls`, you print all files under the current directory; otherwise, print files under that directory. If the argument is not a directory, or doesn't exist, print error message `Not a directory` or `Directory doesn't exist` respectively. You should colorize all directories in the output to green color; all other files use default color. Note you don't need to implement all the flags as in the native command; simply a `ls <directory>` command is enough;
- `find`: you can implement a simple `find` command: `find <directory> <filename>` will find all files with `<filename>` under `<directory>`. This command is very similar to your homework 3! You don't have to implement regex or anything more complicated than what's described above;
- `stat`: same as above; passing an argument to show all the status of the file.

A.2 10 Point Items

- Colorized `ll`: the requirement is the same as `ls`, but you need to produce the same results as in the native commands;

A.3 15 Point Items

- `tree`: this one is harder, since you need to produce a formatted output as seen in `tree` command. Note that you should colorize directories as well (any color you like).

A.4 Note

- You won't be graded for extra credits if you didn't successfully complete the main task, which is the bare minimum;
- If you want to be graded for extra credits, you should comment which command you implemented and where it is in your code, to make grading faster;
- No partial extra credits for each item;
- Extra credits are added to the total of your homework 5, not grand total;
- Unless noted, error handling should be provided in your commands, but how to handle them is up to you, as long as it's reasonable;
- You can get at most 50 extra credits to homework 5.