VS_SHELL

Command Line Interpreter

SAUMYA SHAH 121046

VARSHA TECKCHANDANI 121060

INSTITUTE OF ENGINEERING AND TECHNOLOGY, AHMEDABAD UNIVERSITY

Table of Contents

1. Project overview

Command line interpreter or shell should operate in this basic way:

When one types in a command (in response to its prompt), the shell creates a child process that executes the command entered and then prompts for more user input when it has finished. The shells student implement will be similar to, but much simpler than, the one run every day in Linux. They can find out which shell; they are running by typing echo $SHELL at a prompt. For this project, one do not need to implement much functionality; but one will need to be able to handle running multiple commands simultaneously.

This shell can be run in two ways: interactive and batch.

2. Abstract

1. Batch and Interactive execution

Batch execution involves execution of all the commands in the batch file

For example: If we run a shell script which has a set of commands then all those commands should be serially executed.
Say running shell
[Batch_file_name] will execute the commands in the file "Batch_file_name" until it sees the exit command.

Interactive mode is something similar to the terminal that we use. An interactive console that will allow us to run commands just as one does on the command prompt. Also we need to show proper error messages as and when required. The exit command concept holds in this case too.

2. Built in commands

We will be implementing the basic unix built-in commands such as cd, exit, echo etc

3. Implementation

We have implemented the shell in an in an interactive method. The name of the shell is VS_shell. VS_Shell is a command line interpreter which supports all the basic commands which do not include a change of the directory.

VS_Shell has an additional feature to save the command to some file so that file can be run as a script.

VS_Shell also has options to change the color of the shell.

To make a new file -

Type the command **"newfile"** and then enter the name of the file.

All the commands which are typed now will be saved in that file.

To write a command which should not go in the file-

"command" <space> ;

Example- On writing ls ; the output will be shown but ls command wont be written in the file.

To exit the file-

closefile

To change the color-

Type **"color"** on the VS_shell. Then choose from the given options.

Pipelining-

VS_Shell also supports the pipeline function.

Example- ls | grep shell | wc

4. The ";" operator

When we write a command with the ; operator, and if we are saving the commands in a file, then that particular command will not be updated in the file.

This is done by parsing the command. If at the end of the command, a semicolon is found, the pointer is replaced by a null character and hence the command doesnt get written in the file. If there is no special character and the end of the command, it gets written into the file.

That is done by the strchr() command in c. The strchr() command searches for a particular occurrence.

5. Running Multiple Commands through Pipettes

Vs_shell supports the running of multiple commands through pipettes.

For that, the commands are handled seaprately. There are three flags, input, first

and last.

The command which is entered on the terminal is parsed and it looks for the " | " symbol.

For the first command, input will be 0, first will be 1 and the last will be 0.

Here there is no input as it is the first command. That is stored in a variable. For the second command, the output has to be the input of the first command. So the input flag will have the value which we got for the first command. Similarly, for the third command, the value of the second command shall be passed in last.

It is passed in args[0] and args[1] , the character arrays. The command is saved in args[0].

input: return value from previous command (useful for pipe file descriptor)

first: 1 if first command in pipe-sequence (no input from previous pipe)

last: 1 if last command in pipe-sequence (no input from previous pipe)

For Example:

If you type "ls | grep shell | wc" in VS_shell:

fd1 = command(0, 1, 0), with args[0] = "ls"

fd2 = command(fd1, 0, 0), with args[0] = "grep" and args[1] = "shell"

fd3 = command(fd2, 0, 1), with args[0] = "wc"

So if 'command' returns a file descriptor, the next 'command' has this descriptor as its 'input'.

6. Basic Working

In VS_shell, message parsing is done. The command from the command line is saved in args[0]. It is then passed to the run function. Special commands are checked. Newfile, closefile and color are the commands through which a new file can be made, closefile closes that file and the color command gives the user an option to change the color of the command line.

In the run function, though the logic of the pipettes, the commands are executed. If no pipe is found, then only one command is executed, otherwise multiple commands are executed. The commands are executed using the system call execvp(). In the execvp() , once the command is executed, the program exits. For continuous, interactive interaction, through fork, a child process is created and then that processes exits after the execution which enables the parent process to run till the exit command is given.

7. Basic Pseudo-Code

- Enter the command on the command line interpreter.
- Save the command in an array.
- Check if the input is null or not.
- If not null,
  Check if it's a valid command.
  If yes, Parse the command and execute it.
  Else, print error message
  If null, do nothing
- If | detected in the command Pass the command with input, first and last parameters to the cmd function.
  - Parse for the "|".
  - Assign values for first, input and last accordingly.
  - Run the commands through execvp function.

8. References

http://stackoverflow.com/questions/14301407/how-does-execvp-run-a-command

http://pages.cs.wisc.edu/~swift/classes/cs537-sp09/wiki/pmwiki.php?n=Projects.P1