CS4414 - SPRING 11

Programming Assignment 2 (PA2)

Example Solution

We've installed an executable binary copy of a solution to Assignment 2 on the CSLAB servers (labunix01-03). This solution should meet all of the requirements for the assignment, and also supports full-line comments (i.e., first non-whitespace character on the line is '#'). In addition to the regular executable file, there is a debug version available, plus a couple of sample scripts for you to try.

All files are in the directory `/home/djb4p/cs4414/bin` and should be accessible by everyone. The files are described in Table 1.

File	Description
mysh	Regular (non-debug) version of the shell
mysh_debug	Debugging version of the shell - prints diagnostic messages while processing each input line
test_script0.mysh	A very simple test script
test_script1.mysh	A slightly more complex test script

Table 1: PA2 Example Files

To execute the shell, just type its full pathname, or cd to the cs4414/bin directory and type

```
./mysh
```

You should get a prompt and be able to type Linux commands (e.g., 'ls') and have the shell execute them. To exit from the shell, enter 'quit' or Control-D (i.e., end-of-file). Other commands to try: `cd, pwd, ls -l, cat, man, more, printenv, bash`. You can also try entering some invalid commands like: `set` or `exit`.

To run one of the shell scripts, cd to the cs4414/bin directory and type

```
./mysh script_file
```

Alternatively, you can type

```
./mysh <script_file
```

and the shell will execute the commands in the file but will behave slightly differently.

If you want to use the scripts to test your own shell, just make your own copy of them. If your shell does not support comments, you'll need to edit those out of the script; you may want to leave the blank lines in for readability.

If you haven't implemented the ability to read commands from a file yet, you can still use the scripts for testing by using the second syntax above (i.e., `./mysh <file`), or just manually enter the commands that are in the file.

Also, try setting the shell prompt string. In your login shell,[1] type

```
export MYPS=''somestring''
./mysh
```

[1] For sh, bash, and ksh; for csh type `setenv MYPS ''somestring''`

UPDATE 03/16/2011: NOTE: Your shell should print only the characters in MYPS if it has been set and contains at least one non-whitespace character (i.e., don't add any characters to the contents of MYPS).

To reset the shell prompt, in your login shell[2] type

```
unset MYPS
```

ADDENDUM 03/16/2011

The example solution shell has been modified so that the debugging version prints some extra information. It now prints the current directory both before and after executing the `cd` command, and for the `execvp()` function call it prints the arguments in a different format (i.e., more like the actual argument list).

Also: To see the effect on the `execvp()` call with arguments that are processed internally by the shell (i.e., I/O redirection [<, >, and 2>] and backgrounding [&]), try running a command like this in the debugging version of the shell (mysh_debug):

```
mysh$ ls >/dev/null 2>/dev/null -lR your_home_directory_path &
```

The debug output for the execvp() call should look similar to this:

```
DEBUG: execvp(''ls'',{''ls'',''-lR'',''/home/djb4p'',(nil)})
```

The list inside the curly braces is the argv[] vector that gets passed to the exec call (`(nil)` is the terminating NULL pointer) and the arguments that are processed internally by the shell before executing the user command have been skipped when creating the argv[]. Because the shell is handling the redirection and background arguments itself, they are completely transparent to the user command. Also, redirecting standard output and standard error to `/dev/null` suppresses all output from the `ls` command; redirecting standard input from `/dev/null` is equivalent to reading an empty file - it returns an EOF indication.

ADDENDUM 03/18/2011

In case there's any confusion, your shell program is to be written in C using the C-library (section 3 of the manual) and system calls (section 2 of the manual), and compiled into a single executable binary file using a makefile (as for Assignment 1) on one of the CSLAB Linux machines (labunix01-03). The shell program itself should **not** be (nor include) a shell script nor any C++, C#, Java, Javascript, Objective-C, Perl, PHP, Python, or Ruby code.

The shell's input and output are strictly text and the shell program (mysh) must execute in a character-based terminal emulator or window (e.g., SecureCRT, PuTTY, xterm, Terminal.app, etc.).

ADDENDUM 03/23/2011

When implementing the `cd` command (no arguments), you should change directory to the path given by the HOME environment variable. However, if you type `cd` into your shell followed by `pwd`, you may get a path that's slightly different from what's in the HOME variable (it may have "/cslab" prepended to it, e.g. /cslab/home/djb4p). This is okay as long as the last part of the directory printed by pwd matches what's in your HOME variable and doesn't have anything extra at the end. The additional directory at the beginning

[2]For bash, etc.; for csh type `unsetenv MYPS`

is just a quirk of the way your home directory is mounted from a remote fileserver; other shells like ksh and bash work around this to print the expected path (e.g., /home/djb4p).

Also, when using perror() to print messages for errors that are internal to your shell, you should use "mysh" as the argument except for cases like an error when executing the `cd` command (in which case you'd want to use "cd" as the perror() argument). You can modify the perror() argument to print additional or different information if you like (e.g., the argv[0] of the command that was entered by the user); it's not critical. The main idea is to use perror() to print error messages when `errno` has been set so that the standard error message is printed, and to put something informative in the perror() argument.

Finally, when setting the MYPS variable in your login shell, don't worry if it doesn't seem to take certain characters (e.g., tabs). Just put something reasonable in the variable and have your shell print whatever it's set to (assuming it contains at least one non-whitespace character).