COP5570 Programming Assignment No. 2: FSU Shell (fsh)

PURPOSE

- Practice UNIX system calls
- Practice concurrent programming

DESCRIPTION

In this assignment, you will develop a shell program named FSU shell (fsh), which offers functionality comparable to that of production shells like bash and tcsh. Specifically, fsh will include the following features.

- Execute programs (the basic function of a shell) with or without command line arguments.
- Support the following built-in functions:
  - `set` to set the value of an environment variable. In particular, two built-in enviornment variables, `FSH_PATH` and `prompt` should be supported. `FSH_PATH` is used to record the paths to search for executables. `FSH_PATH` may contain multiple directories separated by character '+'. For example, the command `'set FSH_PATH = /usr/bin+/bin'` sets the variable `FSH_PATH` to contain two directories, `/usr/bin` and */bin*. `Fsh` tries to find a command with the execution permission in all of the directories in `FSH_PATH` when a relative path of the command is specified (the command does not start with '/'). When the executable is not found in the paths, an error message (command not found) should be given to the user. The variable `prompt` is used to set the prompt string for `fsh`. The default prompt should be '<fsh:xxx>' where *xxx* is the count of commands executed in the `fsh` session. The user can set the prompt to be another string. The `set` command can also set the value for other environment variables.
  - `whereis cmd` to show all paths in `FSH_PATH` that lead to `cmd`.
  - `which cmd` to show the path for `cmd`.
  - `show` to show the value of the current `FSH_PATH`.
  - `history #` to show the last # commands executed.
  - `cd dir` to change the current directory to `dir`.
  - `quit` and `exit` to exit `fsh`.
- Support foreground and background execution. Character '&' is used to indicate background execution. Commands without '&' run in foreground. Foreground processes have accesses to keyboard while background processes do not. Keyboard interrupt (typing ctrl-C) will only kill foreground processes, but not background processes (or the shell itself).
- Support I/O redirection. Character '<' is used to redirect the standard input from a file. Character '>' is used to redirect the standard output to a file. For example, 'ls > a' sends to results of `ls` to file `a`.
- Repeat old commands. Character '!' is used to indicate the repeat of an old command. For example, assuming that you have executed three commands 'cp ...', 'more' and 'ls'. A fourth command '!c' will run the 'cp ...' command again.
- Support multiple commands to be executed in one line. Character ';' separates different commands. E.g. 'cat myprog.c; date'.

- Support pipes (up to 5 pipes in a command). Character '|' is used to indicate a pipe. E.g. 'cat myprog.c | grep "int" | more'.
- Support the *up* (↑) and *down* (↓) keys, as in the tcsh (↑ for the previous command, and ↓ for the next command). In `fsh`, 'Ctrl-P' has the same function as the ↑ key, 'Ctrl-N' has the same function as the ↓ key.
- Besides the 'backspace' key, `fsh` also uses 'Ctrl-A' to delete a character in the input.
- `Fsh` should not result in zombie processes.
- `Fsh` should not exit if any of its child processes is still running.
- `Fsh` uses 'Ctrl-K' to kill all child processes created in a section (but not `fsh` itself).

DEADLINES AND MATERIALS TO BE HANDED IN

Feb. 21, 11:59pm. You should clean and tar your project 2 directory and submit on canvas. In the directory, you should have all the needed files to create the executables of the program, the README file and the makefile. A proper makefile should be used in this project: (1) Command 'make'should produce the executable; (2) Command 'make demo' should run the program (use the existing executable if it has been generated or produce the executable if it is not available); (3) Command 'make clean' should remove .o files and executables from the directory; and (4) the program should be compiled with '-Wall -ansi -pedantic' or '-Wall -std=XXX -pedantic' flags. Compiling the program with flags should not have any warning message.

You should also write a README file that includes the basic README file header given in the website and gives information about (1) how to compile and run your program and (2) how to repeat your demo (you will design a demo for your assignment).

GRADING POLICY

A program that cannot compile (with the flags '-Wall -ansi -pedantic' or '-Wall -std=XXX -pedantic') will get 0 point. A program that cannot run any executables is not a shell and will get 0 point. The assignment includes an automatic 5 extra points (105 points out of 100 points).

1. proper README file (5)
2. proper makefile file (5)
3. Run executables(full path) without arguments (5)
4. Run executables(full path) with arguments (5)
5. Built-in functions: show, history, quit, exit (5)
6. Built-in function `set` and built-in variables `FSH_PATH` and `prompt` works properly (e.g Give error messages when the executable is not found in mypath). Built-in functions `whereis`, `which`, and `cd` (5)
7. Background/foreground execution (&) (5)
8. Keyboard interrupt (Ctrl-C) only kill foreground processes (not the `fsh` and not background processes) (5)
9. Backgfound processes have no access to keyboard (5)

10. Multiple(>= 3) commands in one line (;) (5)
11. Repetition of commands (!) (5)
12. File redirection (> and <) (5)
13. Allow one pipe (|) (5)
14. Allow up to 5 pipes (|) (5)
15. ↑, ↓, Ctrl-P and Ctrl-N keys (5)
16. 'backspace' and 'Ctrl-A'keys (5)
17. 'Ctrl-K' key (5)
18. Exit only after all child processes are finished. (5)
19. No zombie processes. (5)
20. A combination of everything. (5)
21. Demo. You must design your demo to show all features in your program WITHIN THE GIVEN TIME (20 minutes). Failing to do so will lose not only the demo points, but also the points for the features that you do not have time to demo. A demo should always start from downloading your submission from canvas. You will not be allowed to modify your source code (unless you are asked to) after the submission deadline. One will lose all 10 points if the source code needs to be modified during demo. (5)
22. Extra 15 points deduction for the second unknown bug of any kind. Extra 8 points deduction for the third known bug or unimplemented feature.
23. 5 points deduction for each warning message in the compilation.

MISCELLANEOUS

- This assignment has more than 1000 lines of code. Please start the project as early as you can.
- You can make any assumptions about the format (syntax) of the commands. The shell should work when the input is correct. (error checking will not be tested).