Operating Systems

Homework Assignment #4

Due 5.5.2016, 23:50

In this assignment, you will use tools you've learned to create a simple shell program. For your shell, learn and use the following: ***execvp*** *(*man **3** execvp, *not to be confused with execv!),* ***pipe, dup, dup2*** (man 2), ***SIGCHLD***.

The program works by reading lines of input from the user:

- Each input line is treated as a command – a program name and its arguments
- Attached to this assignment is a code snippet which reads & parses input lines into an array "*arglist*".
- The *arglist* array is just a split of the input line into words (a word is a non-empty sequence of non-whitespace characters, where whitespace is space, tab, or newline). It is not necessarily the final argument for ***execvp*** (see below).
- Empty lines are detected and ignored (already handled by the code snippet).
- Do not copy this code into your submission! Work on a separate file, compiled **along with** the provided code!

Your shell should follow these guidelines:

- For each user command, execute the relevant process using ***fork*** and ***execvp*** (man **3** execvp, not *execv*).
- The parent process should not handle any further user input until the process finishes.
- If an error occurs in the parent process, terminate it with a proper error message; no need to notify anything to any running child processes. If an error occurs in a child process, output a proper error message and terminate the child process **only**; nothing should change for the parent or other child processes.
- The user command might be invalid! (e.g., a non-existing command/program). This should be treated as an error in the child process (i.e., it must *not terminate the shell*).
- Support background processes
  - If the last argument of the *arglist* array is "&" (ampersand only), run the child process in the background.
  - Do not pass this argument (&) to ***execvp***!
  - The parent should not wait for the child process to finish, but instead continue executing commands.
  - You should prevent zombies and remove them as fast as possible. Do not use threads!
  - Assume background processes don't read input (*stdin*), and that "&" can only appear as the last argument.
- Support nameless pipes
  - If *arglist* contains "|" (pipe only), run several child processes, correctly piping their *stdin* and *stdout* together.
  - You may create several child processes of the original shell, a hierarchy where each process is the parent of the previous one, etc. You're free to choose whatever option so long as it works correctly.
  - To pipe input and output of processes, use ***pipe*** *and* ***dup2*** system calls.
  - Use the same array for all ***execvp*** calls by referencing items in *arglist*, no need to duplicate, malloc, etc.
  - Assume no more than a *single* pipe (|) is provided, and it is correctly placed (at least 1 arg before and after).
- Pipes and background processes will not be combined, i.e., nameless pipes will never be run in the background (however, other background processes might still be running from previous commands).
- Make sure the parent is not interrupted when a child process is running (only the child should terminate)
  - For non-background child processes, make sure only child processes are interrupted by **SIGINT**.
  - Be sure to ignore **SIGINT** correctly (recall that ***fork*** duplicates a process, including its signal handlers).
  - When the child process finishes, restore the parent signal to the previous (**not necessarily default!**) handler.
- You do not need to support special shell commands (e.g., *cd, exit*); only those specified, and executing commands.
- No need to support arguments with quotation marks or apostrophes. Assume these characters are never provided.

<u>Guidelines</u>

- If you are unsure how something should work, base it on the existing terminal and shell in your VM.
  If your shell behaves like the standard shell – it is correct!
- If you're still unsure – ask in the forums.
- Assume the results of the provided parser are correct.
- Error message do not have to be worded exactly as the existing terminal, anything returned by `streror` is ok.
- **As always**, closely follow the forums and all questions & answers provided there. Explanations, guidelines and relaxations may be given there, and they are **<u>mandatory</u>**.

- **Provided code file (*input.c*):**
  - The given code file contains your *main* function. You should create another file (***myshell.c***) implementing the *process_arglist* function.
  - **<u>Do not</u>** modify the given code file, and **<u>do not</u>** submit it. Only use it with your own code (in ***myshell.c***).
  - **<u>Be careful</u>** with the provided code. Recall that you need to free *arglist* and *line* in **<u>all</u>** child processes as well.
  - The original (shell/parent) process should **always** return 1 from *process_arglist*, and **<u>all</u>** child processes should return 0. This will make sure the original process continues processing user commands, while the child processes still free the allocated memory.
  - Recall that on error, you do not have to exit cleanly – you may terminate the child or parent processes (depends on where the error originated) without freeing memory.
- **Submit** a single C file: **myshell.c**
  - **<u>Document it</u>** properly – with a main comment at the beginning of the file, and an explanation for ***every non-trivial*** part of your code. Help the grader understand your solution and the flow of your code.