Project: Write your own shell

The shell is a command interpreter that provides Unix users with an interface to the underlying operating system. It interprets user commands and executes them as independent processes. The shell also allows users to code an interpretive script using simple programming constructs such as if, while and for. With shell scripting, users can coordinate a scenario of their jobs. The behavior of shell simply repeats:

1. Displaying a prompt to indicate that it is ready to accept a next command from its user,

2. Reading a line of keyboard input as a command, and

3. Spawning and having a new process execute the user command.

The prompt symbols frequently used include for example:

cs321%

like a name followed by a % symbol, or simply

$

How does the shell execute a user command? The mechanism follows the steps given below:

1. The shell locates an executable file whose name is specified in the first string given from a keyboard input.

2. It creates a child process by duplicating itself.

3. The duplicated shell overloads its process image with the executable file.

4. The overloaded process receives all the remaining strings given from a keyboard input, and starts a command execution.

For instance, assume that your current working directory includes the a.out executable file and you have typed the following line input from your keyboard.

cs321% a.out a b c

This means that the shell duplicates itself and has this duplicated process execute a.out that receives a, b, and c as its arguments.

The shell has some built-in commands that rather changes its current status than executes a user program. (Note that user programs are distinguished as external commands from the shell built-in commands.) For instance,

cs321% cd public\_html

changes the shell's current working directory to public\_html. Thus, cd is one of the shell built-in command.

The shell can receive two or more commands at a time from a keyboard input. The symbols ';' and '&' are used as a delimiter specifying the end of each single command. If a command is delimited by ';', the shell spawns a new process to execute this command, waits for the process to be terminated, and thereafter continues to interpret the next command. If a command is delimited by '&', the shell goes forward and interprets the next command without waiting for the completion of the current command.

cs321% who & ls & date

executes who, ls, and date concurrently. Note that, if the last command does not have a delimiter, the shell assumes that it is implicitly delimited by ';'. In the above example, the shell waits for the completion of date.

Taking everything in consideration, the more specific behavior of the shell is therefore:

1. Displaying a prompt to show that it is ready to accept a new line input from the keyboard,
2. Reading a keyboard input,
3. Repeating the following interpretation till reaching the end of input line:
   * 1. Changing its current working status if a command is built-in, otherwise
     2. Spawning a new process and having it execute this external command.
     3. Waiting for the process to be terminated if the command is delimited by ';'.

Objective:

In this lab, you will write a simple shell by your own. In this shell, it takes an input command, and then executes it. Your shell should implement the followings:

* Take external commands, such as, ls, emacs and so on.
* Take internal commands
  + **exit**: terminate the shell.
  + **print**: print out the current pid.
  + **help**: help information of your shell.
* Using “&” to implement multiple commands in one line

Descriptions:

* The input to the shell is a sequence of lines. The shell must correctly handle lines of up to 100 characters. If a line containing more than 100 characters is submitted to the shell, it should print some kind of error message and truncate it to 100 characters.
* Each line consists of tokens. Tokens are separated by one or more spaces. A line may contain as many tokens as can fit into 100 characters.
* Words consist of the characters A–Z, a–z, 0–9, dash, dot, forward slash, and underscore. If a word in a line of input to the shell contains any character not in this set, then the shell should print an error message and then ignore the rest.
* Lines of input are divided into token groups. Each token group will result in the shell forking a new process and then executing a program.
* Every token group must begin with a word that is called the command. The words immediately following a command are called arguments and each argument belongs to the command it most closely follows. The order of arguments matters, and they are numbered from left to right, starting at one.

System Calls:

* fork()
* execvp() or execlp()
* wait()
* exit()

Online Reference:

You can refer to an online tutorial below. You can start with this online tutorial.

<https://brennan.io/2015/01/16/write-a-shell-in-c/>

Hand Ins:

Submit 1) source code 2) demo video of the running program. Zip them together and then, upload it on Canvas.