CENG322

PROGRAMMING ASSIGNMENT 2

Due date: 29.04.2024

Write a C program similar to a shell interface.

Your shell-like program displays the command line prompt **`myshell>`** and waits for the user's command. It reads the user's next command, parses it into separate tokens that are used to fill the argument vector for the command to be executed, and executes it. Your shell-like program should support the following built-in commands (which are not part of the regular shell, the functionality is built directly into your shell itself):

Your shell-like program should support the following built-in commands (which are not part of the regular shell, the functionality is built directly into your shell itself):

1) **(15 points) `cd <directory>`** : change the current directory to `<directory>`.
   - Call `chdir()` function.
   - If the argument is not present, change the working directory to the path stored in the $HOME environment variable. You can use `getenv("HOME")` to obtain this.
   - Update PWD environment variable
   - Make sure you support both relative and absolute paths.

2) **(5 points) pwd** : print the current working directory.
   - Simply call `getcwd()` function.

3) **(20 points) `history`** : print the 10 most recently entered commands in your shell in this session, including this command.
   - You are not allowed to use `history` Linux command.
   - You need to maintain a FIFO with a capacity of 10 as your data structure.
   - Note that the commands will be listed in the issue order where each line displays a number from 1 to 10 followed by the command issued. If there are fewer than 10 commands entered, then display all the commands.

4) **(5 points) `exit`** : terminate your shell process.
   - Simply call `exit(0)`.

(15 points and essential for evaluating the tasks below) Support other commands: For other commands, your shell-like program should consider them as system commands. For system commands, your program creates a child process using `fork()` system call, and the child process executes the command by using `execvp()` function. (You can assume that the command does not include a pathname.) You are not allowed to use `system()` function.

(10 points) Support background processes: You need to handle both foreground and background processes for system commands. When a process runs in foreground, your program should wait for the task to complete, then prompt the user for another command. A background process is indicated by placing an ampersand character ('&') at the end of an input line. When a process runs in background, your program should not wait for the task to complete, but display the process id of the background process and immediately prompt the user for another command.

(15 points) Support the pipe operator: For a pipe in the command line, you need to connect `stdout` of the left command to `stdin` of the command following the '|'. For example, if the user types `ls -al | sort`, then the `ls` command is run with `stdout` directed to a Unix pipe, and that the `sort` command is run with `stdin` coming from that same pipe. You don't need to support multiple pipes.

(15 points) Support the logical AND operator: Handle conditionally chained processes using the logical AND operator ('&&'). For example `gcc main.c && ./a.out` must run the executable if and only if the building is successful. This behavior is the result of "short-circuit evaluation." You don't need to support multiple occurrences of this operator.

Code Outline

The outline of your program should be similar to this:

```
while(1)
     print "myshell>"
     read command line
     parse command
     if the command is built-in
          execute command (custom implementation)
     else
          fork a child
          if child
               execute command (execvp)
          else if not background process
               wait for the child
```

Note that this outline does not include a solution to the pipe and logical AND operators. You must come up with a solution to those problems yourself.

Sample Run

Below is a sample run of the desired program:

```
myshell>pwd
/home/std/Desktop
myshell>cd /home/std
myshell>pwd
/home/std
myshell>history
```

```
[1] pwd
[2] cd /home/std
[3] pwd
[4] history
myshell>ls -l
-rw------- 1 std std 152144 Jun 20 2020 alice-in-wonderland.txt
-rw------- 1 std std 13421 Jun 20 2020 calaveras-county.txt
-rw------- 1 std std 635 Jun 20 2021 french.txt
-rw------- 1 std std 172541 Jun 20 2021 looking-glass.txt
drwx------ 14 std std 476 May 25 2021 shakespeare
myshell>gedit &
8550
myshell>kill 8550
myshell>ls -l | wc -l
5
myshell>exit
```

Rules

- You can assume that a command line input will contain at most 100 characters and at most 10 arguments.
- You can assume that all command line arguments will be delimited from other command line arguments by a space character.
- You can assume that at most one of '&', '|' and '&&' is used in an input line.
- Consider all necessary error checking for the programs.
- You need to work individually, no group work is allowed.
- Submit your extensively commented source code via Teams. Please create a compressed file including all source files; and name it as **`yourstudentnumber_P2.zip`** (e.g. If your student number is `201812345678`, the file name must be `201812345678_P2.zip`). If there is only a single C file, then you can submit a file named **`yourstudentnumber_P2.c`** instead.
- No late homework will be accepted.
- You are required to have a demonstration in the following weeks. The exact date will be announced later. Your TA will run and test your program, and ask questions about your implementation. This rule is subject to change.