Nhan Cao
CS 470
Shell Project

Design of the Program

Program functionality:

The C++ program creates a shell for the user which could be used to enter command, view and re-enter command from history.

There are 2 main functions: main() and exec_command(); it also comes with a helper function printHistory(). Depending on user's input, the program might execute the command, show history, retrieve and execute past command, or quit.

If the user choose to execute the command, the program will call exec_command() and pass it the user's input. The function exec_command() will parse user's command and store it in an array. The program (parent) will use fork() to create a child process. It will check to see if the user wants to run the process in the background. If yes, then the parent will use "double fork" to execute it, otherwise, will just fork and exec the command as usual.

The child process will execute the command and destroy itself once finishes. The whole process is not repeated.

To compile this program, enter "make". To run it, enter "./shell".

The structure of history feature:

History is stored in a vector of string of size 10. The index of the command numbered x in the array is

$$(x - 1) \% 10$$

With the indexing set by this algorithm, it is also easy to retrieve the past commands by its number.

Test Plan

The test plan covers 4 features:

- Parsing and executing user input
- Creating and destroying child processes
- Viewing and retrieving history
- Error message

Testing steps after compiling and running the program:

1. "history" – check to see an empty history
2. "!!" – check error message when user trying to retrieve the latest command from the empty history
3. "ls" – see content of the current directory
4. "ls .." – check a command with a argument
5. "ls –a ../.." - check a command with arguments
6. "ps" – check a command
7. "history" – check history feature, now it should only print out 6 past command
8. "skjnd" – checking a non existing command
9. "kjd kdn kdm" – again
10. "top" - check a command
11. "cal"
12. "ls -oneline"
13. "date"
14. "!!" – this should repeat command date
15. "!10" – this should repeat command "ls -oneline"
16. "history" – this would print out the latest 10 commands

Additional Questions

1. **What aspect of process manipulation did you find most difficult to understand?**
   It was the adopting process when you can just abandon a child and the system will just take it in, wait for it until it dies. This was done without any extra specification in the code except the "double fork" technique.
2. **What aspect of process manipulation did you find least difficult to understand?**
   fork() and exec(). A convenient way to use a program to run other programs.
3. **What, if anything, would you change in your current design?**
   Not that I could think of right now.
4. **What, if anything, did you find interesting or surprising about process manipulation that you did not know before doing this project?**
   I took UNIX system before and had some experience using exec(). However, the first time I found out there was a convenient way to run write a program that can run other programs, I was fascinated.