Project No. 1: Quite a Shell (quash)

Submission due: Feb. 29, 2016

PURPOSE

- Getting familiar with the Operating System (UNIX) interface.
- Exercising UNIX system calls.
- Understanding the concept of process from the user point of view.

DESCRIPTION

In this project, you will implement Quite a Shell (quash) using the UNIX system calls. You may work in groups of 2. Quash should behave similar to csh, bash or other popular shell programs. Specifically, the following features should be implemented in quash.

- Quash should be able to run executables (the basic function of a shell) with command line parameters
- If the executable is not specified in the absolute path format (starting with '/'), quash should search the directories in the environment variable PATH (see below). If no executable file is found, quash should print an error message.
- Quash should allow both foreground and background executions. Character '&' is used to indicate background execution. Commands without '&' are assumed to run in foreground
    - When a command is run in the background, quash should print: [JOBID] PID running in background
      example)
      $ program1 &
      [1] 2342
      → job id = 1, pid = 2342
    - When a background command finishes, quash should print:
      [JOBID] PID finished COMMAND
      example)
      [1] 2342 finished program1
- Quash should support the following built-in functions:
    - **set** to set the value of a variable. Quash should support (at least) two built-in variables: `PATH` and `HOME`. `PATH` is used to record the paths to search for executables, while `HOME` points the user's home directory. `PATH` may contain multiple directories (separated by :).
      example)
      `$ set PATH=/usr/bin:/bin`

→ set the variable `PATH` to contain two directories, `/usr/bin and /bin`.
$ set HOME=/home/amir
→ set the user's home directory as '/users/amir'

echo should also be implemented to print the content of the PATH and HOME. The format for echo is as same as bash. As an example

$ echo $PATH
→ /usr/bin:/bin

- **cd** <dir> to change the current working directory to *dir*. if no argument is given, it should change to the directory in the HOME environment variable.
  example)
  $ cd test
  → change the current working directory to ./test
  $ cd
  → change the current working directory to $HOME directory
- **pwd** print the absolute path of the current working directory
  example)
  $ mkdir test
  $ cd test
  $ pwd
  /home/amir/test
- **quit** and **exit** to exit quash.
  example)
  $ quit
  bye
- **jobs** should print all of the currently running background processes in the format: [JOBID] PID COMMAND where JOBID is a unique positive integer quash assigns to the job to identify it, PID is the PID of the child process used for the job, and COMMAND is the command used to invoke the job.
  example)
  $ program1 &
  $ program2 &
  $ jobs
  [1] 2342 program1
  [2] 2343 program2

- Quash should implement I/O redirection. The '<' character is used to redirect the standard input from a file. The '>' character is used to redirect the standard output to a file.
  example)
  `$ ls > a.txt'`
  → store the results of *ls* to file a.txt
- Quash should implement the pipe (|) command.
  example)
  `$ cat myprog.c | more`

- Quash should support reading commands interactively (with a prompt) or reading a set of commands stored in a file that is redirected from standard input.
  example)
  ```
  $ ./quash < commands.txt
  ```

GRADING POLICY

Partial credits will be given for incomplete efforts. However, a program that cannot compile will get 0 points. Point breakdown for features is below:

1. Run executables without arguments (10)
2. Run executables with arguments (10)
3. *set* for HOME and PATH work properly (5)
4. exit and quit work properly (5)
5. *cd* (with and without arguments) and *pwd* works properly (5)
6. PATH works properly. Give error messages when the executable is not found (10)
7. Child process inheritance (inherit environment variable) (10)
8. Allow background/foreground execution (&) (5)
9. Printing/reporting of background processes, (including the jobs command) (10)
10. Allow file redirection (> and <) (10)
11. Allow (1) pipe (|) (10)
12. Report (10)
13. Bonus points
    a. Support multiple pipes in one command. (10)
       $ ls | wc | more
    b. kill command (built-in) delivers signals to background processes. The kill command has the format: `kill SIGNUM JOBID`, where SIGNUM is an integer specifying the signal number, and JOBID is an integer that specifies the job that should receive the signal. (5)
    c. combination of pipe, redirection, and background execution (10)
       $ wc < in.txt > out.txt &

SUBMISSION

Each group should submit the project to your TA via Blackboard. Create a zip file of your code using “make submit”. You should also check that the zipped project still builds and runs correctly after building the submit target with “make unsubmit”. Your TA will be using this command to extract and build your project so make sure it works correctly. If you modify either of these targets, please ensure all file extensions are renamed to “.txt” in the submission and extract correctly with the unsubmit target. The report should describe each of the features in your quash shell and (briefly) how you implemented each feature. Also, describe how you tested quash and document any required features that are not completely implemented in your quash shell.

MISCELLANEOUS

- Start early!
- You need to use C language to implement this project.