Project No. 1: Quite a Shell (quash)
Submission due: October 21, 2020, 2:00pm

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** if you choose. 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. Explore the use of `fork()` and `exec()` to run executables.
- 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. Explore the use of `wait()`, `waitpid()`, and the `SIGCHLD` signal to handle foreground and background processes.
- Quash should support the following built-in functions:
  - *set* to set the value of a variable in the environment. Quash should inherit the initial environment from the command line that invokes it. In C, this is achieved by using the `char **envp` argument to main, as in:

    ```
    int main (int argc, char **argv, char **envp)
    ```

    Quash should support (at least) two built-in variables: PATH, which is used to record the paths to search for executables, and HOME, the user's home directory. PATH may contain multiple directories (separated by :). For example, The command 'set PATH=/usr/bin:/bin' in quash should set the variable PATH to contain two directories, */usr/bin* and */bin*.
  - Child processes should inherit the environment variables (in C, various flavors of the *exec* system call allow you to pass the environment to child processes)
  - *cd* ***dir*** to change the current working directory to `dir`. *cd* with no arguments should change to the directory in the HOME environment variable. Explore the use of `chdir()` to change working directoty.
  - *quit* and *exit* to exit quash.
  - *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.

- When a command is run in the background, quash should print:

  `[JOBID] PID running in background`

- When a background command finishes, quash should print:

  `[JOBID] PID finished COMMAND`

- 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. For example, 'ls > a' sends to results of *ls* to file *a*. Explore the use of `freopen()` to achieve I/O redirection.
- Quash should implement the pipe (|) command. e.g. '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, as in:

  ```
  bash> 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) works properly (5)
6. *PATH* works properly. Give error messages when the executable is not found (10)
7. Child processes inherit the environment (5)
8. Allow background/foreground execution (&) (5)
9. Printing/reporting of background processes, (including the *jobs* command) (10)
10. Allow file redirection (> and <) (5)
11. Allow (1) pipe (|) (10)
12. Supports reading commands from prompt and from file (10)
13. Report (10)
14. Bonus points (you can get bonus points only if you have everything else working (or very close to working))
    - Support multiple pipes in one command. (10)

- *kill* command 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)

MATERIALS TO BE HANDED IN

Only a single person from each group should submit the project *via Blackboard*. Your source file and project report should include the names of all the members of your group. Create a tar file with all of your source code and a Makefile and build instructions. 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

- **You may work in groups of 2** if you choose.
- This project is **not** easy to implement. Please start the project as early as you can.
- All questions related to this project must be entered in the *discussion board* created for this project in the **Projects** page on Blackboard. Ishrak and I will monitor this page regularly. Any questions submitted via an email may go unanswered.
- We strongly recommend that you use C or C++ to implement this project. You may choose a different language, but beware that it may not be possible to implement some of the features in some languages. Also, if you choose to use another language, you may not use any libraries that abstract away the functionality of the shell (for example, libraries that call pipe, fork, and exec in a single routine). Please confirm your choice of language (if different than C or C++) with the class GTA, Ishrak Hayet.