Josh Collins & Shannon Bisges

Project 1: Quash

Report 1

**Required Commands:**

1. Running executable with and without arguments

We parse all commands by getting a char \* buffer using the fgets system call. This limits the amount of text we get from the user so there is no risk of a buffer overflow. We then package all the commands into a vector string which we use to verify the correct number of arguments are given to the program for each command. When actually executing, we utilize execvpe and execve. We use execvpe when executing normal shell commands while execve is used with the direct path and generated executables (aka “./”).

1. Exit/Quit

When running quash, the user should be able to type quit or exit. This will exit the shell and give the user a message letting them know they are terminating quash. This is implemented using exit.

1. Change Directories (cd)

The cd command by itself should change the current path to HOME. When given arguments, cd should change the directory to the path specified. This is implemented by using the chdir command with getenv. Getenv gets the home environment for the chdir command. If chdir is passed a direct path it simply goes there. If it cannot locate the path it throws an error.

1. Background Execution

Executables should be redirected to the background if followed by the '&' symbol. We just check for the ambersand (&) symbol and redirect the output to the background. The job id, pid and command are kept track of in our command class using multiple vectors. The index is consistent between all three. All of the variables are added and taken away concurrently for consistency.

1. Jobs Command

The jobs command should list all of the background jobs that are currently running. We implemented this by entering all jobs that were running in the background into vectors as stated previously. When “jobs” is typed into the command line, a method is called that prints the list of all of the processes in the vectors. When a background job finishes, it is removed from the vectors.

1. File Redirection (Input and Output)

Quash is able to redirect output for any command to a file that it creates using a format similar to “yourcommand > yourfile.txt” this will create the file “yourfile.txt” and place the output of “yourcommand” within the file. In order to redirect the input or output, the dup2 system command is called. We use fopen to get the appropriate input for dup2 using the file and the read/write character.

1. Piping

We create a pid\_t for each command being piped. An integer array is created and then piped. Then the fork command is applied to the first pid. If it is successful, the child will appropriately redirect the output down the pipe before closing it and executing the command. This is then repeated for the second command with the exception of stdin being redirected from the pipe. Once redirection is done and the pipes are closed, waitpid is checked in order to process possible errors that occur.

1. Read commands from File

When running quash, an input file can be fed to it by making the file name the second argument. For example, “./quash input.txt” will run the sample file provided. This contains the commands cd and quit. The command “./quash < input.txt” will yield the same result.

1. Set Home and Path Variables

Home and path are changed by comparing the user input for HOME or PATH and then using the setenv command to change the appropriate variable. This is working correctly.

**Bonus Features:**

1. Kill Command

The kill command can kill a process if given the pid as it's only argument. A user can use the jobs command to find the pid of the program. (This is the second number listed.) By typing “kill pid” the program will kill that process. On the next cycle, quash will tell the user that the process has been terminated. We interpreted jobid to represent the process id or pid.

1. Multiple Pipes

This feature remains unfinished. We were having a small error parsing the code that caused arguments to ls to get added. We think that the implementation is close to finished.

Essentially the function should take in the commands parsed into a vector of strings and run the commands parsing and piping within a for loop that loops for each command within the original pipe command.

**Testing:**

Testing commands are present within the help command. Type help to see testing instructions.

**Compiling and Testing:**

Compile using the make command.

Test using the make test command and alter the input accordingly. Right now there is no input file present. The help command will display the input used to test file input or see the read commands from a file feature discussed earlier in this report.