EECS 678
QUASH
Chris Allmon, Jake Suddock
2/29/16

Our project begins by getting the command from the command line and parsing the line by spaces into an array of arguments held by the command struct. This allows much easier manipulation and analysis of the command passed in. Once parsed, the first word will be checked to match the desired function call with the correct function. Each function/functionality implementation is described below.

SET- This was fairly simple. We simply take the first argument (the variable to change) and set it equal to the second argument (the desired value) using setenv().

EXIT- if the first word in the command is either exit or quit, terminate() is called and the process ends.

PWD- simply call getcwd() and print the result

CD- first, we check that a directory path was passed in. If not, the directory is set to home with chdir. If there is a path passed in, chdir is called with this path as an argument.

EXECUTABLE- We were able to implement the basic executable functionality both with and without arguments. First, the program uses access() to check the passed in directory + file exists. If it does not, the program will go through all paths in $PATH and check for it's existance there. Once access() determines that the file exists in one of these locations, execl is called on it and the file is executed. If the file was not found anywhere, an error is printed to the console.

FILE REDIRECTION- To implement file redirection, we surrounded our executable code inside the child poriton of a fork. When this runs, the child function will run, set the child's standard input and output to whereeever the user specified and then run the executable as normal. Once this finishes, the fork will exit back to the parent, where standard input and output is returned to normal and the program continues to function from the command line. NOTE: unfortunately, when we went to implement executions with arguments, it messed up the redirection and we did not have time to sync the two functionalities.

PIPE- To implement the pipeing mechanism, we get the raw command from the user, then grab the first part of the command and create a fork to run the the entire program based on the single command. Once that is finished, a pipe has been set up and the standard output of the first command is directed to the pipe. Then we return to the parent process where the second part of the command (after the '|') is passed into another fork where the same thing is done, but with the standard input redirected from the pipe created in the previous child process. This format will continue for all the '|' found in the raw command. Unfortunately, we did not have time to get this feature working correctly. There is still a fork/pipe implementation that does not affect regular non piping commands but multiple commands piped together do not.

Unfinished components-
JOBS, Background/foreground execution, Child process inheritance.

Testing:

In order to test Quash thuroughly, each function was created one at a time and was tested thuroughly after each completion. For example, cd was tested with no path, a path from the current directory to both a parent directory and a subdirectory, to a complete path, and an invalid path. Each test was retested after already changing directories to make sure that cd works repeatedly from any directory. After each new feature was added, a more basic set of tests was run on previously existing functions to make sure nothing was affected by the new functions. An example would be after creating set, which deals with PATH and HOME, we would go back and retest cd and execute to make sure that a change to the default PATH and HOME would not affect the functionality of cd and execute.