Introduction

Implementation

The main part of our program is the parsing function. This is where the input string is tokenized and broken into an array of commands. We then check the string for the special characters and specific commands we made our own functions to perform (<, >, &, |, cd, set, quit, exit, and jobs).

File reading (<) was implemented by making a copy of the original input and tokenizing it to separate the filename to read from. The string was then processed to remove new line characters. The process was forked, and an input stream was made and the file read in. The stream was closed after reading was complete.

File writing (>) was implemented by making a copy of the original input and again tokenizeing it to separate the filename to write to. The string was processed to remove new line characters. The process was forked, and an output stream was opened to the file. The original input command was run, and then the output stream was closed; the process exited.

Background Processes (&) are a work in process. This should be implemented by first creating a job, placing it in the background, and then running the command. It is currently unfunctional

Pipes( | ) are implemented by tokenizing the inputCopy string to separate the different commands to run. Two process ids are made and forked. Each process has the whitspace removed before running it through the readCommand function (which is the command that parses the input). This recursive-like setup allows multiple pipes to be fed into the program.

Cd is a function that takes in a string argument of the directory to change to. If no argument is given, it changes the directory to NULL. If an argument is given, it changes to the given directory. There are error handlers to check if the directory is a valid pathname.

Set takes in a string argument that contains the variable to change as well as its arguments. It first tokenizes the string to break up the variable from its arguments. It then sets the environment variable using the setenv function. This function also contains error handling.

Quit and Exit aren't implemented as their own functions; the command line input is parsed, and there is a compare to see if it contains either “quit” or “exit”. If either of these are true, exit(0) is called.

Jobs contains a loop that searches through the jobs array that stores all jobs. Each spot is first checked to see if the job is alive. Then it is printed out in a formatted manner. If no jobs are in the array, it prints a message to the console.

Kill works by taking in an argument of the pid of the job the user wants to kill. It searches for it in the jobs array and then calls the built in kill function on it. If the job couldn't be found, it alerts the user.

Testing

Conclusion