David Qin
PA2: Piping Command Shell
Dr. Ahmed
10/7/18

Parsing

Immediately after reading the input, the program will try to parse by spaces and store them into a vector of strings. It will check for specific cases such as cd or jobs that can't easily be executed by execvp and handle them some other way. If there are no pipes, it will run to a command that has none. If there are pipes or redirects, it will start forking and creating child/parent processes. During this time, the vectors will also be converted into char** arrays, adding a NULL as well, so that execvp needs in order to function.

The function "awk" has a special way of parsing the contents enclosed within the brackets: <>

Piping

There are several cases where the call does not demand piping, in which case just calls run_cmd(), which is my custom function. It will parse by space and convert to string before doing so. This function will start a simple fork that will try to remove the "&" while in the child and then run, acting like a background process while the parent runs it normally afterwards.

Otherwise, it will start a process that can handle one or more pipes. The idea is to keep moving the pointers to other file locations if possible for every pipe. Ex. For the first output, it will be set as the next processes' input and so on. This repeats for all except for the last process, which acts like a normal pipe

Redirection

This will operate within the command that does piping. Basically, it will set the file pointers filein or fileout to whatever file name it reads from the list of arguments. The piping will be set to operate on those respective files over the default pipes given