**PROJECT DESCRIPTION**

myShell runs as a shell program in the terminal. The user can type “./myshell” to run the program as a standard shell, or “./myshell file.sh” to run the contents of file.sh line by line as a batch.

If run as a shell, the user is greeted with the prompt “myshell> “. The user types a command and hits enter, and the shell will run the command. There are 8 builtin in commands, all of which can only be use as the first command of a string of commands (stringing is discussed below):

cd: prints the current working directory. Optionally, the user can supply the directory they would like to switch to.

dir: prints the files of the current working directory. Optionally, the user can supply an alternative directory they would like to print the contents of.

clr: clears the screen.

environ: prints the environment variables created at runtime.

echo: prints whatever follows “echo” on the command line.

help: prints the contents of readme\_doc, which describes how myshell works.

pause: waits for the user to press enter before continuing.

quit: exits myshell

To string commands together:

|: a pipe. In the case of “com1 | com2”, the output of com1 is redirected from the screen to the input of com2.

<, >, >>: redirection. In the case of “com > out.txt”, the output of com is redirected to the file out.txt, which can then be viewed later. “out.txt < com” works the same but is just syntactically reversed, and “com >> out.txt” appends the contents of out.txt with the output of com.

&: background. In the case of “com1 & com2”, com1 and com2 run simultaneously, ie com1 is run in the “background”.

**Mechanism of myshell**

MyShell takes the user input commandline, saved as a string, then separates each substring by trimming the commandline at spaces. The first argument is assumed to be a builtin, an executable command, or a file name. It is stored as the first string in an array of strings in a struct called tokptr. Then, each successive substring of the commandline is stored as an argument to that command by placing it in the string array in the tokptr following the command. This repeats until a redirection flag (<, >>, <, |, &) or the end of the commandline is found. If a redirection flag is found, it is saved in the tokptr’s mode field as a string. The length of the command plus its argument is stored as the tokptr len field. Finally, there is a pointer to the next command tokptr stored in the current tokptr. If the end of the commandline is found, runrecursive() is called to begin running all of the commands stored in the list of tokptrs.

Runrecursive() was designed so that any length of redirection could be run, but I ran into issues that I could no solved in the time frame. Now, myshell essentially can only receive 2 commands/files and their arguments for a single commandline. Runrecursive looks at the mode of the current command, and decides what to do with it. If it is a single command, ie tokptr->next is NULL, the command is run. Otherwise, depending on the redirection mode, it will decide when to create child processes and when to exec, always exiting the command chain if the user tries to use a builtin after the first command.

Piping works by setting up a pipe and a child as the write file and its child as the read file. myshell waits for the first child to end before continuing. Redirection flags work by calling dup2() and changing the stdin/stdout file descriptors to whatever the user has given as the new file. The & operator works by allowing each forked process to run without myshell waiting for execution to complete to return to the prompt.

**Testing**

Testing was done by trying out each of the builtins and creating a short .sh file to run. The one supplied will run, but there are obvious issues with my program, which I will now mention:

1. only 2 commands can be used and redirection can not be used on 2 files, only a command and a corresponding file.
2. piping only works between 2 commands
3. the & operator does work for multiple commands, but sometimes there is a zombie process leftover and I was unable to figure out why. I believe if I did figure it out, I might have been able to solve piping more than 2 commands. For example, if the user types “ps & ps & ps”, all 3 ps commands run simultaneously and a follow up ps shows that only the parent shell and myshell are running. However, if the user types “ls & ls & ls”, all 3 ls commands run simultaneously but a single child is leftover and can be seen as a process “(ls)” when ps is run after. I do not know why this happens with stringed ls commands but not stringed ps commands.
4. If two commands are run in a string, like “ps & ps”, subsequent calls to executable commands are no longer immediately followed by the prompt. Instead, the user must hit enter to see the prompt again.
5. running “ping -c 2 google.com &” causes a segmentation fault somewhere, but the command does run. It however exits at myshell at completion.
6. sometimes, attempt to run executables without arguments still are passed arguments which I cannot see, and I believe the arguments to be NULL. For example, in my .sh file, the first call to ps alone, 3rd line, I give no arguments, but the computer notifies me that I used an illegal argument. Later, the second to last line, I perform ps & ps, and there is no problem.

I believe my main issue is memory allocation in the tokptr structs. I must not be understanding the state they are in as they are created and destroyed. I can demonstrate that free() does not clear the contents of each tokptr or prevent me from accessing their contents after I they have been freed, though this is what I believed free() does. I used memset to clear the contents manually after the commandline is entered. I tried to make sure that every char\* array was NULL terminated and NULL initialized, but there still seems to be something I haven’t thought of yet.