Lab 2 writeup

The purpose of this shell was to parse user input and run any commands the user specified. The shell would wait for input or read it from a file. Depending on what the user specified, the shell would respond accordingly.

Shell pseudocode

If a file was specified: grab input as a line from the file.

If the user will input through the keyboard: grab input and save it into a buffer.

Parse the buffer and tokenize each of the values.

Analyze the values and determine whether the user wishes to perform any redirection, piping, or background execution.

If redirection is specified, properly redirect stdout or stdin and set up any pipes the user wishes to create.

Figure out which command the user wishes to execute.

Check if command is a built-in function the shell provides, if so, execute it.

If not, check if it is a program that is in the same directory as the user, execute it.

If not, check if the program is in one of the system directories, /bin, or /usr/bin and try

Executing from there.

If none of these, report an error.

Print output of program(if any) to the terminal(if no redirection).

Repeat until user types quit.

Testing

I built this program by creating each of the functions and keeping them separate as much as I could before joining them all together at once. This worked out fine for the built-in commands and the parser and analyzer, but things got pretty complicated when I had to tie in the piping and redirection. I had to re-do those methods many times over because they just wouldn’t play along with the rest of the program. Finally I got them to work when I realized I was using the file descriptors wrong and that I had to close any unused ones as I created new pipes. The redirection was clearly the most complex part of the shell. The redirection was working quite nicely on it’s own, but when I had to try to make redirections of both stdin/stdout possible in one command, I had to re-think how to do things. I eventually figured out along with the pipes and the rest was pretty simple. I would create each function, then create a small program to test it with, when I joined these up to the shell program, they worked just as expected.

Methods

Int executeFromFile(char\* file):

Executes the shell and executes each line in file, after which the shell quits.

Int executeFromUser():

Executes a loop waiting for the user to enter a command.

Int redirect(char\*, int):

Enables any I/O redirection if the user indicated with ‘<’, ‘>’, or ‘>>’.

Int resteRedirect():

Called at the end of a cycle to reset I/O to stdin/stdout if they were changed.

Int executeCommand(char\*\*, int):

Loops and sees which command the user wishes to execute.

Checks built-in commands, then any programs from the system.

Executes commands if found.

Int executeProgram(char\*\* command, char \* Dir, int arg, int piped):

Executes a program(the first argument in command), if Dir is not NULL, then it will check in Dir to execute the command, otherwise it will just call exec.

Forks and lets the child execute the command, the parent waits until the child’s completion, unless it is a background process.

If piped is specified, no forking will occur as this is called after the pipe and fork are created.

Int executeLS(char \*\* command):

Prints the contents of the directory command species, if nothing, prints current directory.

Int executeCD(char \* dir):

Attempts to change current directory to dir, if error, reports, otherwise returns 1.

Int findArgsLS(int, char\*\*):

Figures out which arguments user has specified for their ‘dir’ command.

Int checkValidCommand(char \*\* command, int argFlag):

Ensure that the command the user has typed makes sense and is not something that they shouldn’t be allowed to execute.

Void updateCurrentDir():

Updates cwd.

Void IniitaliazeEnvironment():

Called at the the beginning of the shell program and sets the proper environment variables, like hostname, username, and current directory.

char\*\* parseArgument(char\* arg, char\*\* command):

Tokenize arg and place each token into command, alongwith allocating enough space for each argument.

Int freeArgument(char\*\* command, int size):

Called at the end of each cycle to free the current command array.

Int freeCommand(char\* command):

Free a single command and set it’s value to NULL.

Int prepareIO\_BG\_PIPE(char\*\*, int):

Create a pipe between and fork a child whose stdin is redirected to the pipe’s read end.

The parent’s stdout is redirected to the pipe’s write end.

Int analyzeArgument(char\*\* command):

See if command has any redirection symbols in it, if so, create the proper redirections.

Int executeQuit():

Quit shell

Int executeHelp():

Call the help menu

Int executeEnviron():

Print environment strings

Int executeEcho(char\*\* command):

Print command to stdout

Int executePause():

Wait for user to type enter.

Int executeClr():

Clear the terminal screen.

Int generateArgFlag(char\*, int \*, char\*):

Create proper redirections if found.