Lab 2 - Write a Simple Shell – Group 12

Problem Definition

This program is designed to emulate the behavior of a command-line interface by functioning as a simple shell. A shell is an interactive command interpreter that has a set of common commands that are built in. Some of these built-in commands are exiting the shell (*exit*), printing the current working directory (*pwd*), changing the current directory (*cd*), listing directory contents (*ls*), and running external programs.

The first part of the program focuses on interpreting and executing user input. It uses a command parsing mechanism through the *splitCommandLine* function. This function takes the arguments *commandBuffer* (stores user input), *args* (stores individual arguments), and the length of *args* (maximum number of arguments). After finding the words and storing pointers to the beginning of each word, it returns the number of words (number of pointers in the args array).

This function also calls a different function *skipChar*, which skips the character given by the second parameter. When a pointer is passed to the beginning of the array commandBuffer, and the space character (‘ ‘), it returns a pointer to the first ‘t’ character in the array. If the character is null, it returns the original pointer (first parameter) as it was unchanged.

The next part of this program focuses on implementing the internal commands *exit*, *pwd*, *ls*, and *cd*. In the main program, it checks that the number of arguments returned in the previous step is not 0. If true, then call the function *doInternalCommand* which compares the command name entered by the user with the command name from the *cmdStruct* structure. If these command names match, then the shell can execute it.

After executing internal programs, the program focuses on implementing the execution of external programs. In the main program, it checks if number of arguments is greater than 0 and then calls *doInternalCommand* to call *doProgram* if the value returned by *doInternalCommand* is 0. The function *doProgram* loops through each element of the *path* array within the current directory to search for external programs to execute. It uses special features such as *strlen* (find length of current element of path and command) and *malloc* (allocates enough space for the path).

In summary, this shell program efficiently replicates the behavior of a command-line interface, offering easier user interaction with their operating system. The program can efficiently parse and execute user input, handle internal commands, and execute external programs. It also provides informative error messages to enhance the user experience.