CSE 333 - OPERATING SYSTEMS

PROGRAMMING ASSIGNMENT # 2

“MINI SHELL PROGRAM”

ADVISOR:

ASST. PROF. ALİ HAYDAR ÖZER

ADVISEES:

150115502 – BURAK CANİK

150114022 – OĞUZHAN BÖLÜKBAŞ

This programming assignment is related with writing a simple shell by considering the outline program given in the lab sessions. The main() function of our program presents the command line prompt “myshell: ” and then invokes setup() function which waits for the user to enter a command. This program is terminated when the user enters ^D (<CONTROL><D>); and setup function then invokes exit. The contents of the command entered by the user is loaded into the args array.

The shell has some initial commands; bookmark, codesearch, print, set and exit. It supports that a user can enter this commands with required arguments and program runs they without any fork process. In addition to this commands, the user can also use basic Linux commands such that “ls”, “ps”, “gedit” and so on.

The shell supports I/O-redirection on either or both stdin and/or stdout and it can include arguments as well. It can write the standard output of the program to file with overwriting or appending. It can also use the contents of a file as standard input to the program. In addition to that, It can use both of them at the same time, reads the inputs from a file and writes the output to a file.

**Basic lifetime of a shell**

Let’s look at a shell from the top down. A shell does three main things in its lifetime.

* Initialize: In this step, a typical shell would read and execute its configuration files. These change aspects of the shell’s behavior.
* Runs: Next, shell reads commands from stdin and executes them. If command is initial command, directly run it, else fork a child process using fork() from parent process and the child process will invoke execv()
* Terminates: After the shell’s commands are executed, it executes exit command with wants of the user, frees up any memory and terminates but if there is any background process, the parent so the program will wait it until the user close all background processes.

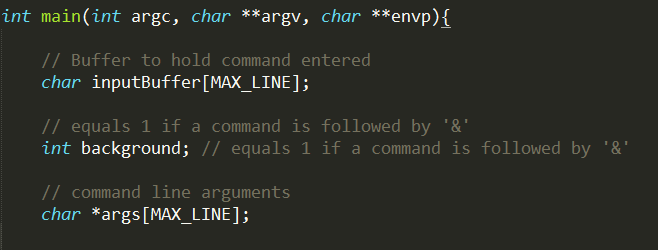
// WARNING! Screenshots of the program codes below is rearranged for clarity.

// They are using to tell main idea and programming logic of the shell program. Do not use any of them!!!

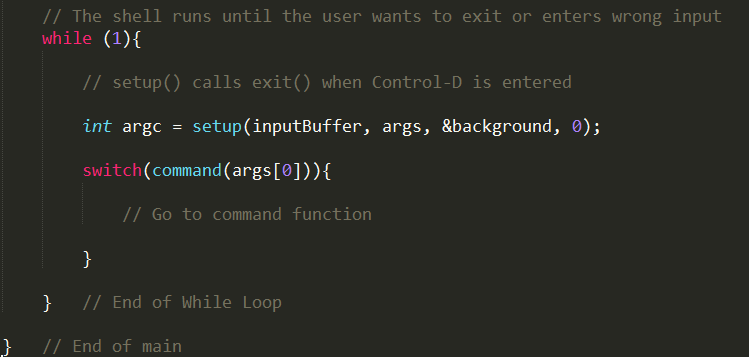
// Comment of them are different from main codes, they are also rearranged

**Main Function**

Main arguments of the shell is below. The envp double pointer holds environmental variables of the system.

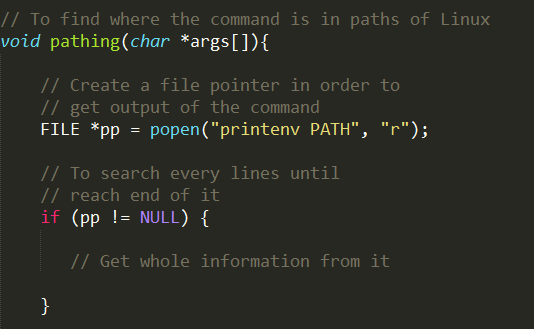


After getting inputs, the shell runs until wrong input or calling exit function. The program divides input into arguments and takes command, argument(s) which is/are exist(s) and any other input(s) if it/they is/are necessary to run program.

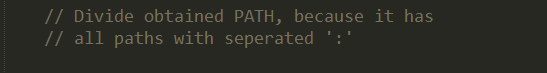


**Path Function**

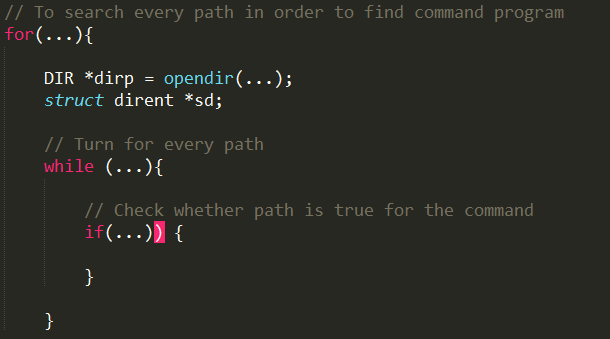
In order to run a command, we must find the path of it. We can do this with using a function, pathing(). It runs printenv command of the Linux in order to find PATH of it and takes it.



Divide the input because it has many paths with separating semicolon ‘:’. It is easy.



After obtaining separate paths, go inside all of them in order to find the command program.



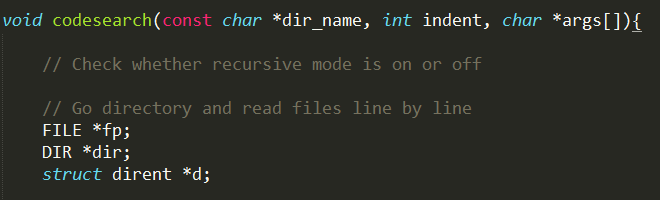
**Bookmark Function**

In bookmark we used a linked list to hold the commands, traversing the linked list when a command at index i is requested. Our linked list also supports removal and insertion as well as printing of the entire list. Using these facilities, we implemented all the necessities of the bookmark command. Also we modified the setup () function slightly so that we can use it to parse complete command strings into arguments when user types “bookmark –i 0” for example. The command string at linked list node 0 is fetched and then its quotation marks are removed using another one of our functions. Then the command string is delivered to setup () function with the last parameter being 1 (meaning “just parse”). Then createChildProcess() is called to fork() and execv () the command with its arguments.

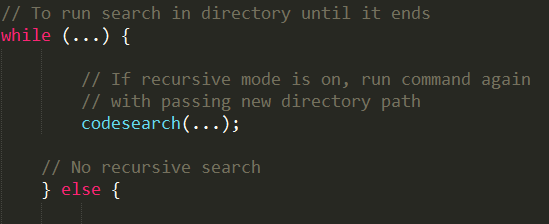
Necessary error checking code was put in place to prevent out of bounds removal and access. Also list will only be printed if there is at least 1 command in it.

**Codesearch Function**

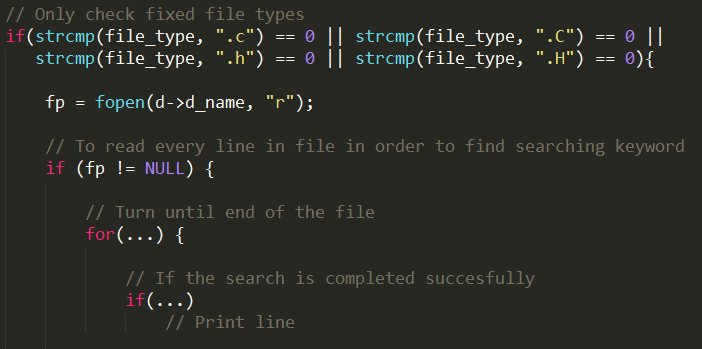
After finding path of command, we can run it. In this function, the shell visits fixed files which are .c, .C, .h, .H. If the user wants to run the program recursively, the shell visits every subdirectories also.



This while loop runs until files and directories are ends. Else part is also used by files in addition to non-recursive mode directory.

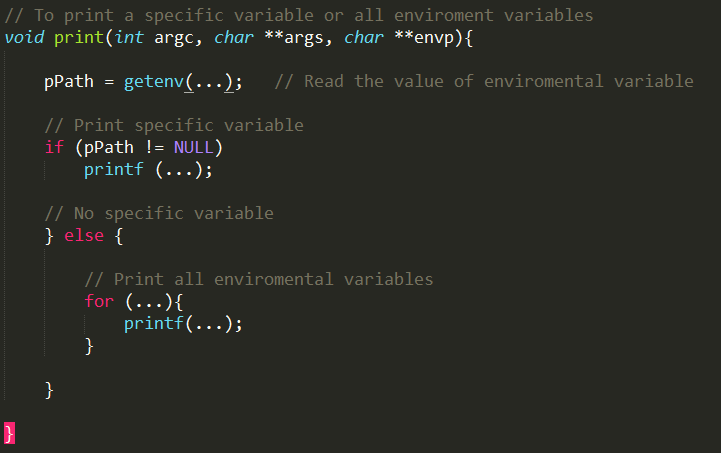


This part checks fixed file types, opens them and searchs every line until end of the file. If the keyword is found, the shell prints it.



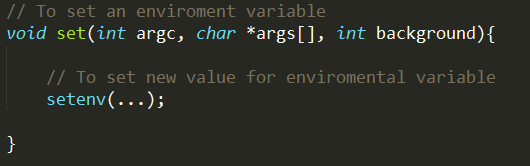
**Print Function**

This function prints a specific environmental variable or all environmental variables if the user does not enter any variable name. gentenv function is very helpful for this function.

****

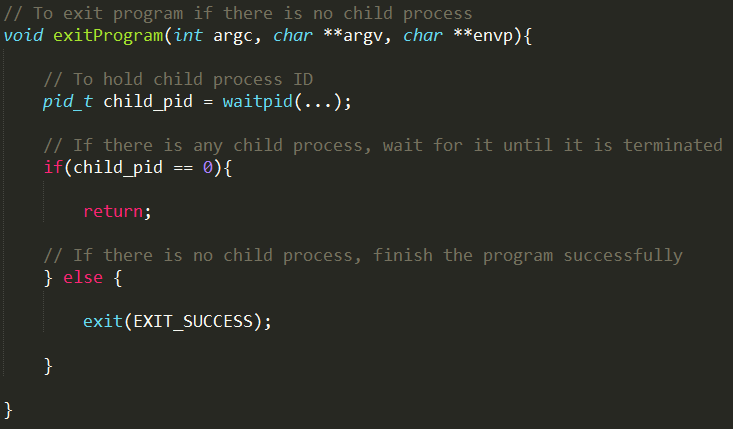
**Set Function**

Maybe the easiest function in this shell is set. Luckily, sentenv does every jobs for us.



**Exit Function**

This function is related the main idea of the shell programming. It does not terminate until the child process(es) is/are terminated. waitpid function check whether there is (a) child process(es) or not. It returns greater than zero if child processes exists, if returns -1 if some error occurred.



**Redirection Part**

Redirection code is pretty straight forward. Parsing the args in createChildProcess (), we determine which kinds of redirections will be applied. We send these redirection types into redirection () which uses a switch construct to determine which redirection types will be applied and calls redirect () to apply them. redirect() is pretty straight forward too. It uses dup2(), open() and close() to implement the necessary redirection. Also necessary error checking/reporting is put in place.