Abraham Schultz

CIS 3207 Temple University

09/19/2019

**Pseudocode for Linux Shell program.**

*// This is the pseudocode for the main function which will contain the main while loop*

*// this is to demonstrate the order which I think the logic should be handled*

***// details for the algorithms I intend to use for each function can be found below the main function’s pseudo code.***

*#define*

*Int 0 = internalCommand*

*Int 1 = externalCommand*

*#define*

*Int 2 = PipeCommand*

*Int 3 = redirectiCommand*

**Main ()** {

// to count and store the users input arguments

Int argc;

Int \*argv;;

// flag to tell if we should execute immediately this will be set to 1 if we detect & at the end of cmd

Int execNow =0;

*// display welcome message and great user*

welcomeMsg ();

*// run program until user selects to exit*

*// main while loop*

While (running == true)

{

*// display prompt which should show working directory*

promtUser ();

Str = getUserInput ();

*// check for redirect symbol*

HandleRedirect();

*//check for piping symbol*

HandlePipe();

// parse input

parseArgs(Str);

*// if command is internal command*

*If (internal command)*

*{*

HandleInternal()

*}else if(!internal command) // else assume this is a file in current directory to open*

*{*

Int result =HandleOpenFile ()

*}*

// otherwise if it was nothing else then assume it was external command

else if (!result)

HandleExternal()

// if all else fails the display error message

*// else std error message*

} // end while

//free memory if needed

} // end main

////////////////////////////////////////////////////////////////////////////FUNCTIONS///////////////////////////////////////////////////////////////////////////////

Int parseArgs(){

//I will use the strok() function to tokenize the input, using a space a delimiter

strok(input,” ”) = token;

// while the string is not null break it apart using white space as delimiter

//also we will push each separate command to the arguments array

while (strok(input,” ”) != NULL) { argc ++; argv[i] = token ; i++}

//- Store the number of strings in the command in the integer variable argc

//- Store the C-Strings in an array of character pointers declared like this: char\* argv[100];

Look at first argv[0] to determine what type of command this is

*If(!arg(0) == list[i] ; I ++)*

If internal type = 0;

Else type =1;

} // end parse args

welcomeMsg () {}; // end welcomeMsg

getUserInput (){

*//getUserInput will make one long string of chars containing everything typed*

use readline to read in user input until user hits enter

}// end getUserInput

promtUser (){ print the name of the working directory followed by any symbol};

*// Enter switch case to handle internal command*

//return 0 if succesful

HandleInternal(){

Switch (internalCommandType)

// quit

Case{

Running = false;

Return 0;

} break

// cd

Case{

//change working directory to input argv[0]

Use the chdir() function and pass it the 1 index in the array of arguments

•If the <directory> argument is not present, print the current directory

•If the specified directory is invalid, generate an error

•This command should also set the PWD environment variable for the shell to <directory>.

} break

// dir

Case{

// display working directory files

} break

// clr

Case{

// this clears the console using escape sequences

// printf("\033[H\033[J")

} break

// echo

Case{

// just reprint argv[0…n]

} break

// help

Case{

// print the user manual using printf()

} break

// pause

Case{

// don’t do anything until user hits enter again

} break

// environ

Case{

// print the environment strings

•Current user

•User’s home path

•Shell name

•OS type

•Hostname

•Directories to search to find an executable.

} break

} // end handleInternal

//return 0 if succesful

// this function will assume that the input given is a file in the local directory and will attempt to open it

**HandleOpenFile ()**{}

//return 0 if succesful

**HandleExternal()** {

// here we will assume that the argv[0] is the first argument in an external command

// we fork then exec the child of the fork giving it arg[0] as the first command, and up to argv[argc]

} // end HandleExternal

**HandlePipe()**;

**HandleRedirect();**

*//this will be the function that gets called to exec any desired external program*

*// it takes the command to be executed along with any arguments to be passed with it*

**execArg(char\* cmd , char\*\* args**)

{

int fork();

// if 0 then exec because we are in new child process

if fork == 0 {

exec(arg[0], argv);

}

// if not 0 then we are in main process ,i.e the shell, and we should check if we need to wait or not

// for child to compete

else {

if (execNow =0)

wait(NULL)

//other wise we don’t wait because we detected &

} // end else

} // end execArgs