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.*

*#define*

*Int 0 = internal command*

*Int 1 = external command*

Main () {

// to count and store the users input arguments

Int argc;

Int \*argv;;

*// 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)*

Int result =HandleInternal()

*// else if command is file in directory*

*Else if (!*result*)*

result =HandleOpenFile ()

// otherwise assume this is an external command

Else if (!result)

HandleExternal()

*// else std error message*

} // end while

} // end main

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

Int parseArgs(){

*//*-Convert each distinct string in the command into a C-String

//- 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];

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

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

If internal type = 0;

Else type =1;

} // end parse args

welcomeMsg () {};

getUserInput (){

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

use readline to read in user input until user hits enter

};

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]

} break

// dir

Case{

// display working directory files

} break

// clr

Case{

// this clears the console

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

} break

// echo

Case{

// just reprint argv[0]

} break

// help

Case{

// print the user manual using pritf()

} break

// pause

Case{

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

} break

// environ

Case{

// print the environment strings

} 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();

execArg(){

// this will be a function that when called forks off from the main program then attempts to exec that new child

// with the given set of arguments

// check if any of the arguments are an & and the parent doesn’t need to wait for the child

}