CSC 60. Fall 2018. Lab 9. **Write your own UNIX Shell, part 1.**

PURPOSE: The purpose of this lab is to allow students to learn a user interface aspect of a UNIX shell. Student will work with process management and some basic system calls.

Important note: please use sp1, sp2, sp3, or atoz servers for this lab.

UNIX Shell

We will start with 3 built-in commands:

- **exit** (exit shell)
- **pwd** (print working directory)
- **cd** (change directory)

The built-in functions are **not** executed by forking and executing an executable. Instead, the shell process executes them itself.

Your shell is basically an interactive loop:

- it repeatedly prints a prompt "**csc60msh >** ",
- parses the input, executes the command specified on that line of input,
- and waits for the command to finish.

FILES TO COPY:

To get the file you need, first move to your class folder by typing: **cd csc60**
Type: **cp -R /gaia/home/faculty/bielr/classfiles_csc60/lab9 .**
Spaces needed: (1) After the **cp** and after the **-R** ↑ Don't miss the space & dot.
(2) After the directory name at the end & before the dot.
After the files are in your account and you are still in **csc60**, you need to type: **chmod 755 lab9**
This will give permissions to the directory.
Next move into lab9 directory by typing: **cd lab9**
Type: **chmod 644 lab9.c**
This will give permissions to the file.
Your new lab9 directory should now contain: lab9.c

Please follow these steps:

- Review the source codes, compile, and execute the programs. Examine the output texts to understand the behavior of each program.
- I have provided you with a simple shell template (**lab9.c**) that you can work from. You build your program from it. **Read** the existing code closely to identify the key components and understand its execution flow. You can compile the shell using the following command: **gcc *lab9.c***
- **Function main.** Handling **built-in Commands**: There are three special cases where your shell should execute a command directly itself instead of running a separate process.
  - *First*, if the user enters **"exit"** as a command, the shell should terminate.
  - *Second*, if the user enters **"cd dir"**, you should change the current directory to "dir" by using the *chdir* system call. If the user simply types **"cd"** (no dir specified), change to the user's home directory. The $HOME environment stores the desired path; use *getenv("HOME")* to obtain this.
  - *Third*, if the user enters **"pwd"**, print the current working directory. This can be obtained with *getcwd()* function.
  - Additionally, we have to deal with the fact that a user might just type an Enter Key with no command.

Pseudo Code (Highlight indicates provided code.)

```
/*----------------------------------------------------------*/
int main (void)
{
    while (TRUE)
    {
        int childPid;
        char *cmdLine;

        print the prompt();     /* i.e. csc60mshell > , Use printf*/

        fgets(cmdline, MAXLINE, stdin);

        /* You have to write the call. The function itself is provided: function parseline */
        Call the function parseline, sending in cmdline & argv, getting back argc

        /* code to print out the argc and the agrv list to make sure it all came in. Required.*/
        Print a line. Ex: "Argc = %i"
        loop starting at zero, thru less than agrc, increment by one.
            print each argv[loop counter]   [("Argv %i = %s \n", i, argv[i]);]

        /* Start processing the built-in commands */
        if ( argc compare equal to zero)
            /* a command was not entered, might have been just Enter or a space&Enter */
            continue to end of while(TRUE)-loop

        // next deal with the built-in commands
        //   Use strcmp to do the test
        //   after each command except for "exit', do a continue to end of while(TRUE)-loop
        if ("exit")
            issue an exit call
        else if ("pwd")
            declare a char variable array of size MAX_PATH_LENGTH to hold the path
                (MAX_PATH_LENGTH is in a #define….look for it)
            do a getcwd (and error check)
            print the path
            continue
        else if ("cd")
            declare a char variable dir as a pointer (with an *)
            if the argc is 1
                use the getenv call with "HOME" and return the value from the call to variable dir
            else
                variable dir gets assigned the value of argv[1]
            execute a call to chdir(dir) with error checking. Message = "error changing directory"
            continue
//.....................IGNORE.....................
//        /* Else, fork off a process */
```

```
//     else {
//             pid = fork();
//         switch(pid)
//             {
//                     case -1:
//                        perror("Shell Program fork error");
//                    exit(EXIT_FAILURE);
//                     case 0:
//                        /* I am child process. I will execute the command, */
//                        /* and call: execvp */
//                        process_input(argc, argv);
//                        break;
//                     default:
//                        /* I am parent process */
//                        if (wait(&status) == -1)
//                                perror("Parent Process error");
//                        else
//                                printf("Child returned status: %d\n",status);
//                        break;
//             }       /* end of the switch */
//...end of the IGNORE above.....................
          }          /* end of the if-else-if */
     }               /* end of the while(TRUE) */
}                    /* end of main */
```

Compilation & Building your program

The use of *gcc* is just fine. If you want to have the output go elsewhere from a.out, type:

gcc *–o name-of-executable* name-of-source-code

Partnership

Students may form a group of 2 students (maximum) to work on this lab. As usual, please always contact your instructor for questions or clarification. Your partner does not have to attend the same section.

Notes for programs with two students

All code files should include both names.

Using **vim**, create a small name file with both of your names in it. When you start your script file, *cat* that name file so both names show up in the script file.

You should BOTH submit your effort. As both of your names occur on everything, when I or another grader find the first submission, we will then give the same grade to the second student.

Hints

Writing your shell in a simple manner is a matter of finding the relevant library routines and calling them properly. Please see the resources section above.

Keep versions of your code. This is in case you need to go back to your older version due to an unforeseen bug/issue.

A lot of code to be used in Lab10 is currently commented out. Leave it there. At the start of Lab10, you will receive instructions about which comment marks you should remove.

Useful Unix System Calls (Also see PowerPoint Slides named **Lab9 Slides**):

getenv/setenv: get/setenv the value of an environment variable
path = **getenv**("PATH");
cwd = **getenv**("PWD");
setenv("PWD", tempbuf, 1);
getcwd: get current working directory.
chdir: change the current working directory (use this to implement **cd**)

C Library functions:

```
#include <string.h>
String compare:
    int strcmp(const char *s1, const char *s2);
    if(strcmp(argv[0], "exit") == 0)      // Sample of complete line.
    strcmp(argv[0], "cd")
    strcmp(argv[0], "pwd")
    strcmp(….,">")
    strcmp(….,"<")

print a system error message:
    perror("Shell Program error");

Input of characters and strings:
    fgets(cmdline, MAXLINE, stdin);
```

Marks Distribution

Lab 9 with the built-in commands is worth 35 points.

Deliverables – This is different from earlier requirements.

Submit **two** files to SacCT**:**

1. lab9.c
2. YourName_Lab9.txt
    - Your program's output test (with various test cases).
    - Please use the UNIX **script** command to capture your program's output.
    - Details below.

Preparing your script file:

Run the program, and enter in sequence:
If you are on a team, cat your name file here.

the Enter Key,
a Space, and then the Enter Key,
pwd
cd ..
pwd
cd lab9
pwd
cd
pwd
exit (exit from the shell)
exit (exit from the script)

If all worked, submit your script file to SacCT. (The script file will NOT contain the contents of lab9.c.)