CSC 60. Fall 2018. Lab 10. **Write your own UNIX Shell, part 2.**

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
In Lab9 we did the 3 built-in commands: cd, pwd, exit.
Now we need to implement: a fork, an exec, and code to handle redirection.

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/files_csc60/lab10 .**
Spaces needed: (1) After the **cp** ↑ Don't miss the space & dot.
(2) After the **-R**
(3) After the directory name at the end & before the dot.
You have now created a lab10 directory and copied in three sample files: execvp.c, redir.c, waitpid.c
Make sure you are still in **csc60,** type: **cp lab9/lab9.c lab10/lab10.c**
We have copied lab9 code and renamed it to lab10.c for you to start work on it.
Next move into **lab10** directory and type: **chmod 644 ***
This will set permissions on the files.
Your new lab10 directory should now contain: lab10.c, waitpid.c, redir.c, and execvp.c

A lot of code to be used in Lab10 is currently commented out.
Use the file **lab9-10 RemoveCommentsGuide.docx** (on Canvas) to guide you to properly remove the extra comments…without removing Every Comment!

Pseudo Code (Yellow highlight indicates the code from Lab9.)

```
/*-------------------------------------------------------*/
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"
```

→ **more on next page**

```
        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, 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
            do a getcwd
            print the path
        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"

        else    /* fork off a process.      This section was commented out for lab9. */
        {
            pid = fork();
            switch(pid)
            {
                case -1:
                    perror("Shell Program fork error");
                    exit(1);
                case 0:
                    /* I am child process.
                    * I will execute the command, call: execvp */
                    process_input(argc, argv);
                    break;
```

→ more on next page

```
            default:
                    /* I am parent process */
                    if (wait(&status) == -1)
                       perror("Shell Program error");
                    else
                       printf("Child returned status: %d\n",status);
                    break;
          } /* end of the switch */
       }      /* end of if-else-if that starts with EXIT
   }        /* end of the while(TRUE)-loop
}           /* end of main
```

---

void process_input (int argc, char **argv)

{

call **handle_redir** passing it argc and argv

call **execvp** passing in argv[0] and argv and return a value to an *integer* variable

(Example: returned_value)

if (returned_value == -1)

error check and do _exit(EXIT_FAILURE)

}

---

void handle_redir(int count, char *agrv[])

{

You need two integer variables to keep track of the location in the string of the redirection symbols, (one for **out_redir** (>), one for **in_redir** (<)). Initialize them to zero.

for loop from 0 to < count

if (“>” == 0) // use strcmp function

if out_redir not equal 0

Cannot output to more than one file. print error. _exit failure.

else if loop_counter compares equal 0

No command entered. print error. _exit failure.

set out_redir to the current loop_counter.

else if (“<” == 0) // use strcmp function

if (in_redir not equal 0)

Cannot input from more than one file. print error. _exit failure.

else if loop_counter compares equal 0

No command entered. print error. _exit failure.

set in_redir to the current loop_counter.

// end of the **if**

// end of the **for** loop

→ **more pseudo code on next page**

```
if(out_redir != 0)
    if argv (indexed by out_redir+1) contains a NULL
        There is no file, so print an error, and _exit in failure.
     Open the file using name from argv, indexed by out_redir+1,
        and assign returned value to fd.  [See 9-Unix, slides 6-10]
        use flags:  to read/write; to create file if needed;
                    to truncate existing file to zero length
        use permission bits for:  user-read; user-write
     Error check the open.  _exit
     Call dup2 to switch standard-out to the value of the file descriptor.
     Close the file
     Set things up for the future exec call by setting argv[out_redir] to NULL
// end of if(out_redir != 0)

if(in_redir != 0)
    if argv (indexed by in_redir+1) contains a NULL
        There is no file, so print an error, and _exit in failure.
    Open the file using name from argv, indexed by in_redir+1
        and assign returned value to fd.  use flags; for read only
     Error check the open. _exit
     Call dup2 to switch standard-in to the value of the file descriptor.
     Close the file
     Set things up for the future exec call by setting argv[in_redir] to NULL
//end of if(in_redir != 0)
```

Resources

Useful Unix System Calls: See PowerPoint Slides file named **Lab10 Slides**

C Library functions:

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

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

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.

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.

Our compiler does not like: for (int i = 0; …..).
It does like it on two lines:

```
int i;
for (i = 0; ……)
```

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.
Use the file **lab9-10 RemoveCommentsGuide.docx** to guide you to remove a set of the extra comments…without deleting Every Comment.

Marks Distribution

Lab 10 is worth 75 points.

Notes for programs with two students

All code files should include both names.
Using **vim**, create a small file with both of your names in it. When you start your script file, *cat* that 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 comments and grade to the second student.

Deliverables

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

1. lab10.c
2. YourName_lab10.txt
    - Your program's output test (with various test cases).
    - Please use the UNIX **script** command to capture your program's output.
    - Details below. (Do not include lab10.c in this file)

Preparing your script file:

Be located in **csc60/lab10** directory.
Run the program, and enter in sequence:

```
    If you are on a team, cat your name file here.
    gcc lab10.c
    Run the program, either a.out or self-chosen name
    ls > lsout              // should work with output going to file
    cat lsout               // display the contents of the output file

    ls > lsout > file1      // should produce an error
    cat foo.txt             // should produce an error
    > lsout                 // should produce an error
    < lsout                 // should produce an error

                            /* wc prints newline, word, and byte counts for each file */
    wc < lsout              // output will go to the screen.
    wc < lsout > wcout      // output will go to a file
    cat wcout               // display the output
    wc < lsout < wcout      // should produce an error

    cd ../lab1              // move to lab1 directory
    gcc lab1.c              // show that the exec works
    a.out                   //show output of lab1
    exit            (exit from the shell)
    exit            (exit from the script)
```

When finished, submit your two files to Canvas. (The script file will NOT contain the contents of lab10.c)