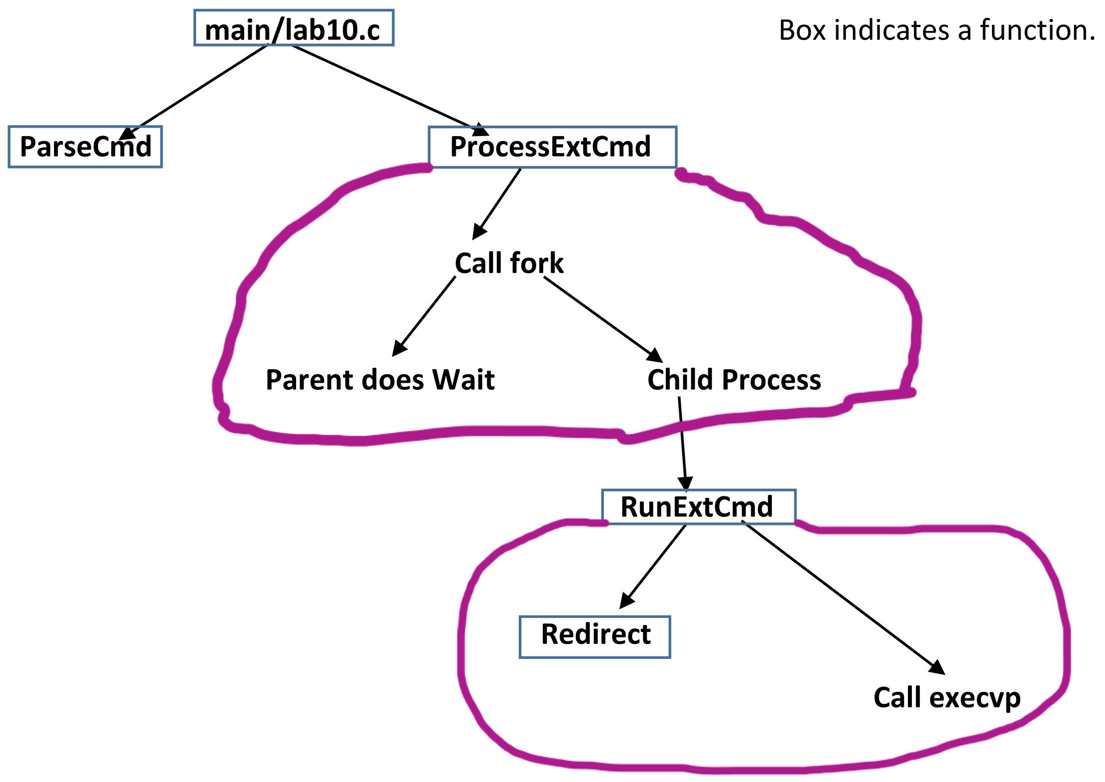
CSC 60. Spring 2023. Lab 10. **Write your own UNIX Shell, part 2.**

Student will work with process management and some basic system calls.

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.

Makeshift Pictorial of the calling sequence of Lab10



FILES TO COPY- Different Instructions this time:

To get the file you need, first move to your class folder by typing: **cd csc60**

Type: **cp -R /home/college/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 Lab10 functions:

~~lab10.c~~, **ParseCmd.c**, **ProcessExtCmd.c**, **lab9_10.h**,

RunExtCmd.c, Redirect.c (These last two are started and you must finish them.)

Stay in directory **csc60**, and you need to

type: **chmod 755 lab10**

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 to **lab10** directory by typing: **cd lab10**
and then type: **chmod 644 ***
This will set permissions on all the files.
Your new lab10 directory should now contain:
lab10.c, **ParseCmd.c**, **ProcessExtCmd.c**, **lab9_10.h**,
RunExtCmd.c, **Redirect.c** (These last two are started and you must finish them.)

EDITING COMMENTS: Some code to be used in Lab10 is currently commented out. You need to remove some comments as directed below.

1. Type: **vim lab10.c**
   Edit the first comment lines to change lab9 to lab10.
   Go to the end of the file. About 4 lines from the end, is the call to *ProcessExtCmd*.
   Remove the // from the beginning of that function call.
   Change the command to be: **`ProcessExtCmd(argc, argv);`**
   Save the file.

2. Type: **vim lab9_10.h**
   The last lines of the file look like this:

```
/* function prototypes */
// NOTE: Function names need to change
int ParseCmd(char *cmdline, char **argv);

//The three function prototypes below will be needed in lab10.
//Leave them here to be used later.
// void ProcessExtCmd(int argc, char **argv);
// void RunExtCmd (int argc, char **argv);
// void Redirect (int argc, char **argv);
```

   Change the lines to look like the 5 lines just below:

```
/* function prototypes */
int ParseCmd(char *cmdline, char **argv);
void ProcessExtCmd(int argc, char **argv);
void RunExtCmd(int argc, char **argv);
void Redirect(int argc, char **argv);
```

   Save the file.

   (Side Note: **char *argv[]** does the same thing as **char **argv**)

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

```
/*----------------------------------------------------------*/
int main (void)
{
    while (TRUE)
    {
        print the prompt();      /* i.e. csc60msh > , Use printf*/
        fgets(cmdline, MAXLINE, stdin);
        /* You have to write the call. The function itself is provided: function parseline */
        Call the function ParseCmd, sending in cmdline & argv, getting back argc

        /* code to print out the argc and the argv list to make sure it all came in. Required.*/
        Print a line. Ex: "Argc = %i"
        loop starting at zero, thru less than argc, 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
             }
            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.  */
        {
                 ProcessExtCmd(argc, argv);    //Note: function name change

        }    /* end of if-else-if that starts with EXIT
    }        /* end of the while(TRUE)-loop
}            /* end of main
```

```
void ProcessExtCmd(int argc, char ** argv)    //PROVIDED TO YOU
{         int status;
          pid_t pid:

          pid = fork();
          switch(pid)
          {
             case -1:
                  perror("Fork error");
                  exit(EXIT_FAILURE);
             case 0:
                  // I am the child process. I will execute the call to execvp
                  RunExtCmd(argc, argv);      //updated function name
                  break;
             default:
                  // I am the parent process.
                  if(wait(&status) == -1)
                      perror (Error on the parent wait);
                  else
                      printf("Child returned status: %d\n", status);
                  break;
          } /* end of the switch */
         return;
}
```

```
void RunExtCmd (int argc, char **argv)       // You add code to complete this function.
                    // Child Process
                    // It will be in its own file.
{
     call Redirect passing it argc and argv      //(Don't need any * on argv)

     call execvp passing in argv[0] and argv and return a value to an integer variable
                                                        (Example of int variable:  ret)
     if (ret == -1)
     {
          error check and  _exit(EXIT_FAILURE)
     }
}
```

More on next page →

---

void Redirect(int argc, char *argv[]) *//You write this function in its own file*
*//Include the file **lab9_10.h**. There is a starter file.*

```
    // This function is a Child Process
    //You need two integer variables to keep track of the location in the string of the
    // redirection symbols, (one for out (>), one for in (<) ). Initialize them to zero.
    // When I use the words loop or loop counter, I am referring to whatever variable (i) you use
    // in your loop.
{                                                                  grey is legal; yellow is illegal
    for loop from 0 to < argc
        if ( “>” == 0)          // use strcmp function                  ls > lsout > file1
            if out not equal 0                                          0  1  2     3  4
                Cannot output to more than one file.  fprintf error. _exit failure.
            else if loop_counter compares equal 0                       ??       >  lsout
                No command entered. fprintf error. _exit failure.                0    1
            set out to the current loop_counter.
      else if (“<” == 0)        // use strcmp function                  wc < lsout < file1
            if in not equal 0                                           0   1  2     3  4
                Cannot input from more than one file.  fprintf error. _exit failure.
            else if the loop_counter compares equal 0
                No command entered. fprintf error. _exit failure.       ?? < lsout
            set in to the current loop_counter.                            0    1
      // end of the if -else if
    // end of the for loop

    if(out != 0)
        if argv (indexed by out +1) contains a NULL                     ls >  NULL
            There is no file, so fprintf an error, and _exit in failure. 0  1  ??
        Open the file using name from argv, indexed by out+1,                ls > lsout
            and assign returned value to fd.  [See 9-Unix, slides 6-10]      0  1  2
                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. perror & _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] to NULL
    // end of if(out != 0)
                                                                        wc < lsout > wcout
    if(in != 0)
        if argv (indexed by in +1) contains a NULL
            There is no file, so fprintf an error, and _exit in failure.
        Open the file using name from argv, indexed by in+1
            and assign returned value to fd.  Use flag; for read only
        Error check the open. perror & _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] to NULL
    //end of if(in != 0)
```

Word of warning: In the past many students have duplicated the code for the OpenOutputFile section to be used for OpenInputFile section. If you do that, lots of little things need to be changed. Be careful.

Resources

Useful Unix System Calls: Also see PowerPoint Slides file named **Lab10 Slides**

C Library functions:

```
#include <string.h>
String compare:
   int strcmp(const char *s1, const char *s2); //Function prototype from string.h

   if(strcmp(argv[0], "exit") == 0)      //Sample. One line completed.
   strcmp(argv[0],"pwd")
   strcmp(argv[0],"cd")
   strcmp(….,">")                        //Sample.  if(strcmp(argv[i], ">") == 0)
   strcmp(….,"<")

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

Hints

Our compiler does not like: **for (int i = 0; …..)**

You will receive the following errors:

```
        test_loopcounter.c:6: error: 'for' loop initial declarations are only allowed in C99 mode
        test_loopcounter.c:6: note: use option -std=c99 or -std=gnu99 to compile your code
```

These errors imply that on every "gcc" line, you must add: -std=c99 OR -std=gnu99.

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.

Compilation & Building your program

You need to write a makefile to pull all the program files and functions together.
The function Redirect.c must be in its own file, and NOT in lab9_10.h

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.

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.
<u>You must BOTH submit the same code and script</u>. Each of these files will include both of your names.

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.

Preparing your script file:

Be in **csc60/lab10** directory. When all is well and correct,
type: **script lab10.txt**

If you are on a team, **cat** ***your name file here****.*

At the prompt,
type: **touch lab9_10.h**
type: **make**
type: **lab10** to run the program (or whatever name you used)

Enter in sequence:

```
1.  ls > lsout          // should work with output going to file
2.  cat lsout           // display the contents of the output file

3.  ls > lsout > file1  // should produce an error
5.  cat foo.txt         // should produce an error
6.  > lsout             // should produce an error
7.  < lsout             // should produce an error
```

*// **wc** prints newline, word, and byte counts for each file*

```
8.  wc < lsout          // output will go to the screen.
```

```
 9.  wc < lsout > wcout        // output will go to a file
10.  cat wcout                 // display the output
11.  wc < lsout < wcout        // should produce an error

12.  cd ../lab2          // move to lab2 directory
13.  pwd                 // show that we landed in lab2
14.  gcc lab2.c          // show that the exec works
15.  a.out               // OR ./a.out   Show output of lab2
16.  exit                // (exit from the shell)
17.  exit                // (exit from the script)
```

Points Value

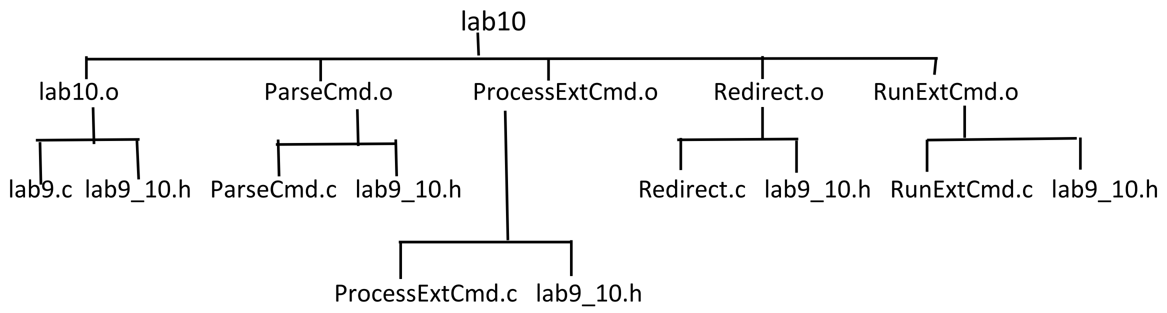
Lab 10 is worth 86 points.

Deliverables

Submit **six** files to Canvas**. Please no zip files on Canvas.**

1. makefile
2. lab9_10.h
3. lab10.c
4. RunExtCmd.c
5. Redirect.c
6. lab10.txt (the script file)

Dependency Chart for the makefile (added late by request)


lab10
lab10.o
ParseCmd.o
ProcessExtCmd.o
Redirect.o
RunExtCmd.o
lab9.c
lab9_10.h
ParseCmd.c
lab9_10.h
Redirect.c
lab9_10.h
RunExtCmd.c
lab9_10.h
ProcessExtCmd.c
lab9_10.h