CSC 2223 - Systems Programming
TechShell Programming Project

Write a C program (called **techshell.c**) that repeatedly allows a user to enter input (interpreted as a shell command). Your program should first parse the command (i.e., entered as text) and determine whether it contains an I/O redirection (< for an input from a file, > for an output to a file). The command is then executed in a forked child process that uses the **execvp** command to execute. Your program should provide an informative shell prompt that contains the current working directory, followed by a $, and ending with a space. The prompt should be updated if the current working directory changes. Here's an example that illustrates this:

```
/home/zak$ cd ..
/home$
```

You must include a seperate file called **README.txt** that provides the following details:

1. Your name (and partners' name if applicable);
2. The inner workings of your program; and
3. If you have not fully implemented the shell functionality as described above, list parts that work (and how to test them if it's not obvious) so that you can receive partial credit.

You must also include output from testing your program (separate from the README) using the command "**script**", this is a program that records terminal sessions. See the sample below (red is main shell):

```
~$ script
Script started, file is typescript
~$ ./techshell
$ cd Desktop
Desktop$ exit
~$ exit
exit
Script done, file is typescript
~$
```

A **script** session of your program is started (and saved in a file called **typescript**) by typing script from the command line, running your program, testing it, exiting it, and finally exiting script, as seen above.

Make sure to implement good programming practices. For example, your **main** function should merely be a driver (i.e., use various functions and subroutines).

Submit a single **.zip** file that contains the following:

1. The README file described above;
2. All **.c** source files required to compile your program (do not include any executable or object files);
3. Output from testing your program via **script** command, make sure to demonstrate the following:
   (1) Simple UNIX commands;
   (2) I/O redirection (< and >); and
   (3) Error conditions (e.g., invalid commands, errors, etc).

***Note:**

- In order to receive a grade for this assignment, **you need to demo your work to the instructor.** A schedule will be released closer towards the end of the quarter (i.e., usually 2 weeks before the quarter ends).
- You may work in groups of **three** for this project.
- Do **NOT** use the **system(...)** or **popen** functions

***Hints:**

(1) To get the value of an environment variable, **getenv** is your friend;

(2) The **execvp** is the required function to complete this program correctly;

(3) The following may just work to redirect **stdin** from a file:

```
// you may want to look into the dup2 function
FILE* infile = fopen(my_input_file, "r");
dup2(fileno(infile), 0);
fclose(infile);
```

(4) The following may just work to redirect **stdout** to a file:

```
FILE* outfile = fopen(my_output_file, "w");
dup2(fileno(outfile), 1);
fclose(outfile);
```

(5) To build command line arguments, **malloc** and **realloc** are your friend;

(6) To copy strings, you may want to try **strdup**;

(7) To handle errors, **errno** and **strerror** are your friends; and

(8) **#define DEBUG** may greatly help with debugging (i.e., allows for toggle-able print statements).

Use the template file as a good starting point. Make sure to comment your source code appropriately, and to include a header providing your name(s).

Simplified Grading Rubric:

- Your program compiles with no errors (50 pts)
- I/O redirections work (50 pts)
- All other commands handled through the **execvp** function (60 pts)
- Invalid user input handled correctly (20 pts)
- **README** file provided (10 pts)
- **typescript** file provided (10 pts)
- Demo completed (100 pts)

Your program should be able to handle the following examples (via simple **script** output). Note that your program will be tested with more than these commands (including more error handling cases):

```
~$ ./techshell
~$ cd Desktop
Desktop$ pwd
/home/zak/Desktop
Desktop$ cd ..
$ pwd
/home/zak
$ ls -lh techshell.c
-rw-r–r– 1 zak zak 3.9K Oct 27 15:08 techshell.c
$ chmod 400 techshell.c
$ ls -lh techshell.c
-r——— 1 zak zak 3.9K Oct 27 15:08 techshell.c
$ ls /root
ls: cannot open directory /root: Permission denied
$ horseface
Error 2 (No such file or directory)
$ cd /root
Error 13 (Permission denied)
$ ps
PID TTY TIME CMD
6271 pts/3 00:00:00 bash
7415 pts/3 00:00:00 techshell
7460 pts/3 00:00:00 ps
$ whereis ps
ps: /bin/ps /usr/share/man/man1/ps.1.gz
```

```
$ /bin/ps u
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
zak     444     0.0     0.0    7236 5244 pts/1 Ss+ 07:57 0:01 bash
zak     6271    0.0     0.0    7220 5032 pts/3 Ss 14:18 0:00 bash
zak     7415    0.0     0.0    2172 1324 pts/3 S+ 15:08 0:00 ./techshell
zak     7483    0.0     0.0    5228 2372 pts/3 R+ 15:12 0:00 ps u
$ ls
Desktop Documents Music techshell Downloads Pictures techshell.c
Videos
$ ps u > ps.out
$ wc -l < ps.out
5
$ wc -l < ps.out > wc.out
$ cat wc.out
5
$ ls
Desktop Documents Music techshell Downloads Pictures techshell.c
Videos ps.out wc.out
$ rm ps.out wc.out
$ ls
Desktop Documents Music techshell Downloads Pictures techshell.c
Videos
$ exit
~$
```