Homework: shell

This assignment will make you more familiar with the Unix system call interface and the shell by implementing several features in a small shell, which we will refer to as the 6.828 shell. You can do this assignment on any operating system that supports the Unix API (a Linux Athena machine, your laptop with Linux or MacOS, etc.). Submit your 6.828 shell to the submission web site as a text file with the name "hwN.c", where N is the homework number as listed on the schedule.

Read Chapter 0 of the xv6 book.

If you are not familiar with what a shell does, do the Unix hands-on from 6.033.

Download the 6.828 shell, and look it over. The 6.828 shell contains two main parts: parsing shell commands and implementing them. The parser recognizes only simple shell commands such as the following:

```
ls > y
cat < y | sort | uniq | wc > y1
cat y1
rm y1
ls |  sort | uniq | wc
rm y
```

Cut and paste these commands into a file `t.sh`

To compile `sh.c`, you need a C compiler, such as gcc. On Athena, you can type:

```
$ add gnu
```

to make gcc available. If you are using your own computer, you may have to install gcc.

Once you have a gcc, you can compile the skeleton shell as follows:

```
$ gcc sh.c
```

which produce an `a.out` file, which you can run:

```
$ ./a.out < t.sh
```

This execution will print error messages because you have not implemented several features. In the rest of this assignment you will implement those features.

Executing simple commands

Implement simple commands, such as:

```
$ ls
```

The parser already builds an `execcmd` for you, so the only code you have to write is for the `' '` case in `runcmd`. You might find it useful to look at the manual page for exec; type "man 3 exec", and read about `execv`. Print an error message when exec fails.

To test your program, compile and run the resulting a.out:

```
6.828$./a.out
```

This prints a prompt and waits for input. `sh.c` prints as prompt `6.828$` so that you don't get confused with your computer's shell. Now type to your shell:

```
6.828$ ls
```

Your shell should print an error message (unless there is a program named `ls` in your working directory). Now type to your shell:

```
6.828$ /bin/ls
```

This should execute the program `/bin/ls`, which should print out the file names in your working directory. You can stop the 6.828 shell by tying ctrl-d, which should put you back in your computer's shell.

You may want to change the 6.828 shell to always try `/bin`, if the program doesn't exist in the current working directory, so that below you don't have to type "/bin" for each program. If you are ambitious you can implement support for a `PATH` variable.

I/O redirection

Implement I/O redirection commands so that you can run:

```
echo "6.828 is cool" > x.txt
cat < x.txt
```

The parser already recognizes ">" and "<", and builds a `redircmd` for you, so your job is just filling out the missing code in `runcmd` for those symbols. You might find the man pages for open and close useful

Make sure you print an error message if one of the system calls you are using fails.

Make sure your implementation runs correctly with the above test input. A common error is to forget to specify the permission with which the file must be created (i.e., the 3rd argument to open).

Implement pipes

Implement pipes so that you can run command pipelines such as:

```
$ ls | sort | uniq | wc
```

The parser already recognizes "|", and builds a `pipecmd` for you, so the only code you must write is for the '|' case in `runcmd`. You might find the man pages for pipe, fork, close, and dup useful.

Test that you can run the above pipeline. The `sort` program may be in the directory `/usr/bin/` and in that case you can type the absolute pathname `/usr/bin/sort` to run sort. (In your computer's shell you can type `which sort` to find out which directory in the shell's search path has an executable named "sort".)

Now you should be able the following command correctly:

```
6.828$ a.out < t.sh
```

Make sure you use the right absolute pathnames for the programs.

Don't forget to submit your solution to the submission web site, with or without challenge solutions.

Challenge exercises

You can add any feature of your choice to your shell. But, you may want consider the following as a start:

- Implement lists of commands, separated by ";"
- Implement sub shells by implementing "(" and ")"
- Implement running commands in the background by supporting "&" and "wait"

All of these require making changing to the parser and the `runcmd` function.