***Due Date:*** *February 29, 2020, 11:59 pm*

**Objectives:**

* Gain insight into how an OS shell functions.
* Expand your understanding that an “OS is just a program.”
* Begin to understand the amount of complexity in writing the source code for an OS

**Pre-Work**

This homework is a continuation of the lab05. If you haven’t finished the steps **1 to 3**, please do it before you start working on this homework.

**Problem Description:**

1. After you have placed sh.c into a directory on the Vulcan server, you can compile it with the command gcc sh.c. This will produce an output file named a.out, which you can observe by listing the contents of the directory with a ls command.
2. Once you have created a.out, you can even run the minimal shell with the command ./a.out. While it will run, it won’t do very much because several critical commands have not been implemented (yet). You can tell you are inside the minimal shell because the command line prompt has been replaced by 6.828$. Inside this little shell, enter <ctrl>+d to stop it and return to the regular Linux prompt.
3. Now create a new file named t.sh and paste these lines into it:

ls > y

cat < y | sort | uniq | wc > y1

cat y1

rm y1

ls | sort | uniq | wc

rm y

then save the file. Next, enter the command ./a.out < t.sh and you should get a series of error messages. This confirms you commands such as ls, cat, and so forth don’t work.

1. Your assignment is to write the code to a) implement simple commands, b) perform I/O redirection, and c) implement pipes. You will do this by opening the source code for sh.c in an editor, writing some code, saving and recompiling the file, then running ./a.out again.

## Executing simple commands

1. 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. Now type to your shell:

6.828$ ls

If this works, it will list all of the files in your current directory. (If it does not work, try entering 6.828$ /bin/ls instead. This should work; if it does, you may want to change the sh.c shell to always try /bin, so that below you don’t have to type “/bin” for each program.)

**I/O redirection**

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

echo "learning OSes 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.

Note that the mode field in redircmd contains access modes (e.g., O\_RDONLY), which you should pass in the flags argument to open; see parseredirs for the mode values that the shell is using and the manual page for open for the flags argument.

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

1. 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”.) Make sure you use the right absolute pathnames for the programs.

Now you should be able to run the following command correctly:

6.828$ a.out < t.sh

which did not work at the outset.

1. Once you are able to do so, take a screenshot of running a.out < t.sh
2. Okay; that this point, we’ve built and installed XV6, been introduced to writing and compiling C program code, learned how to use the debugger to investigate the status inside of XV6 while it’s running, and had a first taste of modifying a shell by rewriting the C source code. Now we are all set to begin modifying the XV6 system as we have received it.
3. Write up your homework report. You should explain each step that you did along the way, including what it means to getting the operating system running.
   1. Discuss your understanding of a command line shell. Explain how simple everyday C code can be implemented to make all of the functionality that a command line OS shell can perform.
   2. Include in your homework report the results of running a.out < t.sh before sh.c was fixed up and a copy of what it looked like when you were all finished.
   3. Comment on the homework objectives and the degree to which you feel that you have accomplished them. Feel free to reflect on the learning you’ve done in this lab/hw and in the labs/hws of the semester as a whole so far. Thoughtful reflection looking back over an experience is an excellent way to organize your thoughts and helps set the experience into memory better.

When complete, submit the homework report in the link given in Canvas by the due date.

**Notes:**

* You are required to turn in a written report as a document (.doc, .docx, or .pdf file) of the hw2 to Canvas.
* You can log in to the CIS Department Vulcan Linux servers using moat.cis.uab.edu from anywhere, including off campus. From on campus only, you may elect to log in instead to vulcanNN.cis.uab.edu, where NN is a number from 10 to 19. Doing this avoids the moat load-balancing software, which sometimes has difficulties. However, you have no control over whether the server you choose is already quite busy, or not. Using moat eliminates this issue.