CS360 Jsh Lab

- James S. Plank
- CS360
- Url: http://web.eecs.utk.edu/~jplank/jplank/classes/cs360/360/labs/Lab-8-Jsh/index.html

---

What you turn in

You should submit the program **jsh.c**. The TA's will compile this with **libfdr**.

Introduction

Time to write a shell. Your job is to write **jsh** -- your own shell, which is going to be very primitive. **Jsh** is like **csh / sh / bash**: It is a command line interpreter that lets you execute commands and redirects their input/output. You will use the same syntax as the other shells shell, and implement the operations < , >, >>, & and |. You are not expected to deal with things like quotes, environment variable expansions, pattern matching, job control or command histories.

I recommend doing this in three parts:

Part 1

Part 1 is to implement a really bare-bones shell with no frills. This shell should take one optional command line argument, which is its prompt. If not specified, the prompt should be **"jsh: "**. If the prompt is "-", then do not print any prompt.

Jsh reads lines from standard input, and if they are non-blank, it attempts to execute them. Like the shell, **jsh** should search the **PATH** variable to find executable files specified as relative path names (in other words, **jsh** just needs to call **execvp** instead of **execve**). If a command line ends with an ampersand (&), then **jsh** should not wait for the command to finish before returning the prompt. Otherwise, it should wait for the command to finish.

Jsh should exit when the user types **CNTL-D** or **exit**.

Don't worry about **CNTL-C** or **CNTL-Z**. However, when you quit, look around and kill stray processes that might have been left around by **jsh** while you are debugging it.

Also, don't worry about file indirection, which is covered in the next part. Just get your **fork()**'s, **wait()**'s and **execvp()**'s correct.

You may want to look at **forkcat1.c** and **forkcat2.c** for some examples of using **fork()**, **wait()** and **execvp()**.

Remember Dr. Plank's Cardinal Sin of Exec!

Also, read below about zombie processes and make sure you have that taken care of before continuing onto part 2.

Part 2

Here you add input and output redirection to files -- i.e. <, >, and >>. This should also be straightforward from the Dup lecture. You may assume that the tokens <, >, and >> (and & and | for that matter) are separated from

other arguments by white space (i.e. you may use the fields library).

Part 3

Now, implement pipes. In other words, any combination of |, <, >, and >> and & should work. Look at **headsort.c** for some pointers on connecting two processes together with a pipe.

Be careful that you wait for all processes in a pipe to complete before going on (that is, unless you have specified with & that you don't want to wait).

Just to reiterate, you can have *any* number of processes piped together. For example, the following should work (it will reverse the file **f1** and put the result in **f2**):

```
jsh: cat -n f1 | sort -nr | sed s/...../// | cat > f2
```

General Stuff

Flush before fork

Before you call fork, you should call **fflush()** on stdin, stdout and stderr. Trust me.

Zombies

You should try to minimize the number of zombie processes that will exist (this is in all parts). This is not to say that they can't exist for a little while, but not forever. When you call **wait()** for a shell command, it might return the pid of a zombie process, and not the process you thought would return. This is fine --- you just have to be able to deal with it. (i.e. consider the following sequence):

```
jsh: cat f1 > /dev/null &
jsh: vi lab3.c
```

You are going to call **wait()** to wait for the **vi** command to terminate, but it will return with the status of the zombie process from the **cat**. This is all fine -- you just need to be aware that these things may happen, and that you may have to call **wait()** again to wait for **vi** to complete.

Open files

You must make sure that when you call **execvp**, that there are only three files open -- 0, 1, and 2. If there are others open, then you have a bug in your shell.

Also, when a command is done, and the shell prints out its prompt, then it should only have three files open -- 0, 1, and 2. Otherwise, you have forgotten to close a file descriptor or two and have a bug in your code. Check for this. My **jsh** never uses a file descriptor higher than 5.

Waiting in jsh

In **jsh**, if you do not specify the ampersand, then your shell should not continue until *all* the processes in the pipe have completed. You'll need a red-black tree for this.

Errors

Your code should work in the face of errors. For example, if you specify a bad output file at the end of a multi-stage pipe, then the error should be noted, and your shell should continue working. Make sure you check for all

the error conditions that you can think of.

The Grading Script -- Helper Programs

The first 20 gradescripts test part 1. The next 40 gradescripts test part 2, and the remaining ones test part 3.

The following programs are in the lab directory:

- **cattostde.c**: This works like **cat**, but it puts standard input to standard error.
- **strays.c**: This checks for open file descriptors and will flag an error if any file descriptor higher than three is open. Then it works just like **cat**
- **strays-files.c**: This works like **strays** except it copies the first argument to the second.
- **strays-fsleep.c**: This works like **strays-files** except it sleeps for a 5th of a second before starting.
- **strays-sleep.c**: This works like **strays** except it sleeps for a 5th of a second before starting.

The grading script uses all of these to test various features of your shells. Beside the first few, each gradescript call will take between 1 and 20 seconds. The gradescripts are time sensitive, too, so the output of your program may change as time passes -- for that reason, the gradescripts can be a little hard to parse. I can go over them in class if you ask me. I've written up an example of going through one of the difficult ones (gradescript 61) in http://web.eecs.utk.edu/~jplank/jplank/classes/cs360/360/labs/Lab-8-Jsh/gs-61.html.