← Lab 7: Processes and Error handling

Due by midnight, Wednesday 11/7

For project 4, you'll need to be able to start new processes. Error handling is also really important, and will prevent you from accidentally creating a forkbomb.

Getting started

Right click and download this link. This is the skeleton of your lab.

Please be sure to put your name at the top.

This program takes any number of command-line arguments and **runs the program specified.** I've already written the `fork()`, `execvp()` and `waitpid()` system calls for you, but you've got some work to do to **make it output some useful error messages.**

For example, if you compile it and run it like so:

```
$ ./lab7 ls -l
total 17
-rwxr-xr-x 1 abc123 UNKNOWN1 6875 Apr  6 02:04 lab7
-rw-r--r-- 1 abc123 UNKNOWN1 1593 Apr  6 02:02 lab7.c
----------
$ _
```

Right now, all it does is run the program, and then print a line of `--------`.

This program can serve as the basis of your project 4, since the main responsibility of a shell is to start new processes and report errors from them!

What to do

Read the comments in the code I've given you to see what you have to do and where to do it.

This lab is also testing to see if you can follow those instructions by looking up documentation, so **use the `man` pages.**

`man` pages are... dense. They have a lot of information, and they go into a lot of detail. But the important parts to focus on are:

- The first paragraph of the "DESCRIPTION" as well as any parts of it that explain the arguments
- The "RETURN VALUE" section
- Sometimes, the "ERRORS" sections

Here are some pages you will find useful:

- `man 3 perror`
  - *the 3 makes it look up the C library call instead of some MySQL thing.*
  - *if the manpages you find make no sense, try putting the 3 in there.*
- `man 3 exit`
- `man execve`
- `man signal`
  - this will tell you how to **ignore a kind of signal** with the signal() function.
- `man waitpid`
  - will tell you about how to check the return value and status from the process.
  - the `WIFEXITED(status)` stuff is kinda weird, so here's an example on how to use those functions:

```
// this is code that's already in the lab7.c file for you.
int status;
int childpid = waitpid(-1, &status, 0);

if(WIFEXITED(status))
        // it exited normally; use WEXITSTATUS() to extract th

// et cetera...
```

Things to test

Try running some simple commands that you know will complete successfully. Your output might look like:

```
$ ./lab7 ls
lab7  lab7.c
----------
Program exited successfully!

$ ./lab7 echo "hello"
hello
----------
Program exited successfully!

$ _
```

Then try running a **valid command that will fail,** like trying to `ls` a directory that doesn't exist:

```
$ ./lab7 ls /bogus
ls: cannot access '/bogus': No such file or directory
----------
Program exited with error code 2

$ _
```

Then try running some **bogus commands** that should cause the `execvp` to fail. In addition to printing the message about the invalid command, you should *also* see a report about the error code - that should be **the error code that you pass to the `exit()` after `execvp()` failed.**

```
$ ./lab7 aisfjajojojedge
Error running program: No such file or directory
----------
Program exited with error code 1

$ _
```

Then, try killing a process with a signal. An easy one to use is `SIGINT`, which is sent when you use `ctrl+C`. That's why you set up the `SIGINT` handler to be ignored. For example:

```
$ ./lab7 cat
hello
hello
^C----------
```

```
Program was terminated by signal: Interrupt

$ _
```

This happens because `cat` without arguments simply copies `stdin` to `stdout`, so we got trapped in it. Pressing `ctrl+C` sends `SIGINT` to `cat`, which kills it, which *your* program can detect and print an error about.

Submission

Please remove all the comments I put in the file before you submit.

Then submit as usual.

© 2016-2018 Jarrett Billingsley