CS333 Spring 2016

Lab 4 (Take Home)

Goal

The goal of this assignment is to give you a basic introduction to the OS shell. You will run a few simple commands in the bash shell and understand how it works.

Before you start

- Review the `proc` filesystem and the `ps` command.
- Read through Chapter 0 in the xv6 textbook, especially the description about how the xv6 shell works.

Exercises

Do the following exercises, and record your observations in your report.

1. Open a bash shell. Find its `pid`. Write down the process tree starting from the first `init` process (`pid = 1`) to your bash shell, and describe how you obtained it. You may want to use the `pstree` command.

2. Consider the following commands that you can type in the bash shell: `cd`, `ls`, `history`, `ps`. Which of these are system programs that are simply exec'ed by the bash shell, and which are implemented by the bash code itself?

3. Take the `cpu1print` code from Lab 1. Run the following command in bash.

   ```
   $./cpu1print > /tmp/tmp.txt &
   ```

   Find out the pid of the new process. Go to the `proc` folder of this process, and describe where its I/O file descriptors 0, 1, 2 are pointing to. Can you describe how I/O redirection is being implemented by bash?

4. Run the following command with `cpu1print`.

   ```
   $./cpu1print  | grep hello
   ```

Once again, identify which processes are spawned by bash, look at the file descriptor information in their `proc` folders, and use it to explain how pipes work in bash.

5. The next few exercises will lay the groundwork for your next lab. Modify the server from Lab 2 to write a program called `server-slow.c`. This program will simply sleep for 1 second after every write into the socket. This exercise may not make much sense to you now, but it will be very useful for your debugging during the next lab. For many tasks in the next lab, you will need your file downloaded to last a very long time, so that you can perform some interesting tests (like signal handling) during the download. With your slow server, you can make downloads of even 2MB files to last a long time, so that you can easily debug.

6. Next, write a simple standalone client for the file server. Write a program called `get-one-file.c` that takes four arguments: the file name to fetch, the server IP address, server port number, and a variable to indicate whether the client should display the downloaded file contents on stdout or not. Your client can be a simple single threaded client that just sends one request to fetch one file and exits. (You should have already written something like this during your initial phases of building the client in Lab 2.) You may test it with the server of Lab 2 (or your slow server above). Below are shown two invocations of the client program to get a file from a server running on the local host at port 5000. You can see the use of the two options: `display` and `nodisplay`.

```
$./get-one-file files/foo100.txt 127.0.0.1 5000 nodisplay
$./get-one-file files/foo100.txt 127.0.0.1 5000 display
... < downloaded file contents displayed to standard output> ...
```

7. Next, write a signal handler to handle the SIGINT (Ctrl+C) signal in your client. Normally, when you type Ctrl+C, the default signal handler provided by the OS will cause the `get-one-file` process to terminate without printing anything. You must now write `get-one-file-sig.c` that has a signal handler to handle SIGINT. This signal handler should print out how many bytes have been downloaded before terminating the process, as shown below. (Note how running with a slow server is useful to debug this client, as you will have ample time to hit Ctrl+C and observe the effect.)

```
$./get-one-file-sig files/foo100.txt 127.0.0.1 5000 nodisplay
Received SIGINT; downloaded 52037 bytes so far.
```

Submission and Grading

You may solve this assignment in groups of one or two students. You must submit a tar gzipped file, whose filename is a string of the roll numbers of your group members separated by an underscore. For example, `rollnumber1_rollnumber2.tgz`. The tar file should contain the following:

- `report.pdf`, which contains your answers to the exercises above. Be clear and concise in your writing.
- Your code `server-slow.c`, `get-one-file.c`, `get-one-file-sig.c`. Please make sure your code is well documented and readable.

- An optional makefile to build your code.

Evaluation of this lab will be as follows.

- We will run your code and check that it compiles and runs correctly, handles signals correctly, and so on.
- We will read your report to check your understanding of concepts.