Lab 08: Server services

Hands-on Unix system administration DeCal

2012-03-12, due 2012-03-19

Final project

OCF account

An OCF account is required, if you still do not have one, please let us know and request one.

Proposals

Final project proposals are **due in hard copy at the start of next class**, March 19 (after spring break). Please arrange to stick around after the lecture portion of class is complete that day — we'll be meeting with each group to discuss your proposals during lab. Your proposal should include the following:

- **Elevator pitch.** Convince me that your project is awesome. What will it do, and why should I care? What problem will it solve, or how will it make my life as a Unix system administrator easier?
- **Summary.** How do you plan to implement your project? Will there be multiple interfaces, server side or client side, or documentation?
- **Timeline.** List major milestones necessary to complete your project, how long each should take, and by when you plan to have finished each milestone. You'll be presenting your project to the class on 25 April, so plan accordingly.

Bonus points[1] if your proposal looks professional (hint: if you're a CS or math major, you really should learn how to use LaTeX).

DNS queries

If you're doing this exercise on an Instructional server, note that `host` and `dig` are installed but are not in your `PATH`, otherwise use the OCF login server, `ssh.ocf.berkeley.edu`.

1. Use `host` to look up the `MX` record for `berkeley.edu`. Show the syntax you used.
2. What's an NS record, and what are `berkeley.edu`'s NS records? Where are the servers `berkeley.edu`'s NS records point to actually located, and why might IS&T have set things up that way?
3. Use `dig` to perform the same queries as in questions (1) and (2), and show the syntax you used. Then run `dig` with no arguments — what do you see in the "ANSWER SECTION"? What are these servers?

[1]Well, not really — this is a pass/not-pass class, after all. But you *do* want to be awesome, don't you?

HTTP

If you're doing this exercise on your own machine, you'll want to install `netcat` (`nc`), otherwise use the OCF login server, `ssh.ocf.berkeley.edu`.

In this part, you'll be writing a simple shell script that downloads the contents of an arbitrary URL, saving the result to the file "`output`." Assume that the URL your script is passed is in the format

```
http://www.example.com/path/to/document
```

(don't worry about escapes or special characters). Here's what your script should do:

1. Use `cut` to extract the domain name (`www.example.com`) and the path (`/path/to/...`). Bear in mind that a valid path can contain an arbitrary number of slashes.
2. Use `nc` to connect to the server on port 80. (You can pipe input into netcat's stdin.) An HTTP request looks like this:

   ```
   GET /path/to/document HTTP/1.1 Host: www.example.com (newline)
   ```

   Use `echo` and the "enable interpretation of backslash escapes" option — your HTTP request, on one line, will look something like

   ```
   GET /path/to/document HTTP/1.1\ nHost:  www.example.com\ n\ n
   ```

   (note the *two* concluding newlines).
3. Pipe this input into a read loop. We're not interested in the HTTP headers, which are terminated by a blank line, so you should discard lines until you find an empty one. (To check if you've found an empty line, test whether it's equal to the output of `echo -e '\ r'` — try comparing it to `$'\ r'`.) Once you've found a blank line, `echo` every remaining line.
4. Save the output of this read loop to a file called "`output`." Please save trees — don't print any output in your lab hard copy.

Before you begin, try out `nc` and `echo -e` on their own and make sure you understand how they work.

Extra for Experts™!

Once you're done, if you're feeling adventurous, you might try...

- The filename "`output`" is not very descriptive. Make your script save its output somewhere more useful.
- Look at the Wikipedia article on HTTP status codes. The first HTTP header your script receives includes a status code — common ones include "`200 OK`," "`301 Moved Permanently`," "`404 Not Found`," and so on. Build in support for a few of these status codes — maybe follow redirects (look at the `Location:` header), or print a warning when you encounter a `404`.
- If you're a shell-scripting ninja, you finished the entire lab in ten minutes flat, and you're bored, read the appropriate RFCs and write a `bash`-based IRC client.