Programming Project 3: HTTP

Summary In this assignment, you will write and use a simple HTTP client, server, and proxy.

Deadline Friday, June 7, 2013, 10:00pm

Rules of Engagement

- For this project, the Python packages `httplib`, `urllib`, `urllib2`, `BaseHTTPServer`, `SimpleHTTPServer`, and `CGIHTTPServer` are off limits.
- You may work together in **teams of two** on the programming portions.
- The project report is to be completed individually.

Milestone 1. HTTP Client (20 points)

You will begin by creating a simple client program. Start with the example program at:
`http://wiki.python.org/moin/TcpCommunication#Client`

Requirements For Milestone 1, your client must:

- Extract the hostname and URL from the command line
- Perform a DNS lookup to retrieve the IP address for the server using `socket.gethostbyname()`
- Initiate a TCP connection to the server's IP address, port 80, using `socket.socket`
- Submit a valid HTTP 1.1 request for the desired URL
- Parse the return code from the response
- If the request was successful ("200 OK"), read the entire data from the server and save the returned object to a file with the name given on the command line. Your client must support retrieving files of more than a few KB, which require multiple calls to your socket's `recv()` method.
- The command-line syntax for running your program will be:

  ```
  python client.py <URL> <filename>
  ```

  For example, to fetch `http://www.cs.pdx.edu/index.html`,

  ```
  python client.py http://www.cs.pdx.edu/index.html cs.html
  ```

Deliverables: `client.py`

Milestone 2. HTTP Server (20 points)

Next, you will build a very simple web server. Start with the example code at:
`http://wiki.python.org/moin/TcpCommunication#Server`

Requirements For Milestone 2, your server must:

- Listen on the TCP port specified on the command line. The program will be run as

  ```
  python server.py -p <port>
  ```

- Serve files from its local directory. So, for example, if you run the server program from its own local directory (e.g. "python server.py -p 8000"), the request

  ```
  GET /server.py HTTP/1.1
  ```

  should retrieve the source code for your program.

- Send a valid HTTP response, including headers and the complete file data
- Return status code "200 OK" for files that exist
- Return status code "404 File Not Found" for filenames that don't exist
- Support retrieving files larger than a few KB, requiring multiple calls to your socket's `send()` method

You may find the Python package `os.path` useful for translating URL paths into local directory paths (`os.path.join()`) and for determining whether or not a requested file exists (`os.path.exists()`).

Deliverables: `server.py`

Milestone 3. HTTP Proxy (20 points)

You will then combine elements of your HTTP client and server to create a simple HTTP proxy.

Requirements For Milestone 3, your proxy must:

- Listen on the TCP port specified on the command line. As above, the program will be run as

  ```
  python proxy.py -p <port>
  ```

- Parse each request to extract the server name (from the `Host:` header) and the requested URL path. You may assume that the Host header will always be present.
- Look up the IP address for the given server
- Initiate a TCP connection to the server's IP address on port 80
- Copy the HTTP request to the server
- Retrieve the HTTP response from the server, and forward it on to the client. Your proxy must support responses larger than a few KB, which require multiple calls to the sockets' `recv()` and `send()` methods.
- Finally, your proxy must keep a log of each HTTP request that it serves. The log should be a text file, named `log.txt`, with each request represented by one line in the file. The format for the log is:

  ```
  server path IP
  ```

  So for example, if you retrieve `http://www.example.com/index.html` from IP address 1.2.3.4, the line in your log file would be:

  ```
  www.example.com /index.html 1.2.3.4
  ```

Deliverables: `proxy.py`

Milestone 4. Experiments and Project Report (20 points)

Your report should answer the following questions:

1. Use a web browser to load the course web page through your proxy. The URL is `http://www.cs.pdx.edu/~cvwright/courses/2013/494-594/`
   (a) How many IP addresses did your proxy contact?
   (b) How many domain names did your proxy contact?
   (c) How many requests did your browser make?
2. Use a web browser to load `http://www.google.com/` through your proxy.
   (a) How many IP addresses did your proxy contact?
   (b) How many domain names did your proxy contact?
   (c) How many requests did your browser make?
3. Use a web browser to load `http://www.cnn.com/` through your proxy.
   (a) How many IP addresses did your proxy contact?

   (b) How many domain names did your proxy contact?
   (c) How many requests did your browser make?

4. Approximately how much of your code from the client and the server were you able to re-use for your proxy?
5. What worked well for you in this project?
6. If you had to do this again, what would you do differently?
7. What was your role on your team? Which parts of the code did you write?
8. What was your partner's role on your team? Which parts of the code did they write?

Deliverables: `report.{txt/doc/pdf}`