COMP 112 Assignment 1: HTTP Servers

Lead TA: Ashley Hedberg
Based on an assignment from Alva Couch
Tufts University

Due 11:59 PM February 9, 2015

Introduction

In this assignment, you will write a web server that understands the HTTP protocol. Your server will respond to two URLs: one which displays a list of browsers that have recently loaded the page, and one which displays information about you. These pages will not correspond to files, but will be constructed by your program in response to each request.

Requirements

Write a program a1.c that does the following:

- Takes a single argument: a port number on which to listen for requests.
- Accepts TCP requests from web browsers and responds with a single HTML page of content based on the URL:
  - The main page (/): Descriptions of the last 10 browsers that accessed it.
  - An about page (about.html): Anything you’d like to say about yourself. It doesn’t have to be long.
- Protects users from script injection attacks.

The requirement for the main page is satisfied by keeping records in your server about the identities of the browsers that send you requests. You do
not need to open files or databases; you can do this in memory. Restarting your server will cause it to clear its memory, and that’s fine.

The key to this assignment is to keep things simple and to utilize your understanding of the TCP and HTTP protocols.

**HTTP and HTML Responses**

Every HTTP request is a stream of characters, separated by \n characters, which describe parts of the request. Many of these characters describe the browser that is asking for information.

Similarly, your response should be a stream of characters. The simplest HTTP response is:

```
HTTP/1.1 200 OK
Content-type: text/plain

hello world
```

The first line is the protocol description, which defines how your program intends to communicate in the subsequent lines. The second line is the content type, which says you will respond in plain text. The blank line is a delimiter. It is required between the header and the payload. Technically, there are two \n characters here. The **hello world** is the text that the browser should display.

To serve HTML, this pattern needs to be modified slightly:

```
HTTP/1.1 200 OK
Content-type: text/html

<html><body><h1>hello world</h1></body></html>
```

Please note: we are not looking for beautiful web pages. Do not spend time trying to properly return CSS in your response.

**Responding to Multiple URLs**

While keeping your program self-contained, it should respond to two different URLs with different results. One will provide browser statistics, while
the other will provide a page about you. Link from each page to the other, using URLs local to your server. Completing this will require interpreting the HTTP GET portion of the request.

Script Injection Attacks

It is not a safe assumption that the “browser” that contacts your server is a traditional web browser. It could be a malicious program that sends intentionally corrupt headers for malicious reasons. One easy way for a malicious person to wreck your assignment is to contact your server and embed malicious HTML and/or JavaScript into the HTTP request headers (where it does not belong). If you display these headers as if they were text, your server won’t be affected, but it can compromise and/or crash the browser of anyone visiting your server afterwards.

The best way to prevent this is to replace several “dangerous” HTML special characters that may appear in the request headers with their print equivalents:

<table>
<thead>
<tr>
<th>Character</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td><code>&amp;lt;</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&amp;gt;</code></td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td><code>&amp;quot;</code></td>
</tr>
</tbody>
</table>

In other words, if you receive a request header containing `<`, your server’s output should contain `&lt;` instead. This makes it impossible for scripts to run on your webpage.

Where to Work

Because you are writing a server, you may not work on your project from the homework server. Inside the departmental firewall (i.e. from a machine in Halligan, or from a public-facing server), please ssh to one of the following machines:

- comp112-01.cs.tufts.edu
- comp112-02.cs.tufts.edu
- comp112-03.cs.tufts.edu
- comp112-04.cs.tufts.edu
- comp112-05.cs.tufts.edu
You may only work on your server from these machines.

As you will learn in this course, every service runs on a port—a number between 1 and 32767. You will be assigned ports that are yours alone. To receive your port assignments, use the /comp/112/bin/ports command, which was written by Alva Couch.

```
~ > /comp/112/bin/ports
ahedbe01, your ports are 9020-9039
PLEASE BE SURE TO WORK ONLY ON COMP112 MACHINES. THANKS!
~ >
```

When you test your services, you must stay within your own port range. Two services cannot share a port, so if you steal someone else’s port, their service will behave strangely. Please be kind to your classmates, and do not steal their ports!

**Testing**

The servers on which you will write your code are protected from the outside world by a firewall. Your server will not be visible from the public internet. To test your server, you must do so from a point within the Halligan network.

You server will call functions from the network services library. When compiling, you will need to link against this library with the `-lns1` flag:

```
gcc -g a1.c -lnsl
```

Start your program as follows, replacing 9020 with one of your port numbers:

```
./a.out 9020
```

At this point, your service is running. To see if it’s working, open a browser on any Halligan machine and visit http://comp112-01.cs.tufts.edu:9020. (This is assuming you’re on the comp112-01 server.) This should connect to your program and display the list of browsers that have recently accessed your page. Similarly, http://comp112-01.cs.tufts.edu:9020/about.html should display information about you.

If you are connecting remotely, you can also use the command-line `wget` and `curl` to test your server.
Submission and Lateness Policy

Submit your code by using provide: `provide comp112 a1 a1.c`

Late submissions will be accepted up to three days past the deadline, with a penalty of 10% per day late. For circumstances that you believe warrant an additional extension, speak to Prof. Dogar.