Proxy Recitation

Jeffrey Liu
Recitation 13: November 23, 2015
Outline

- **Getting content on the web: Telnet/cURL Demo**
  - How the web really works
- **Networking Basics**
- **Proxy**
  - Due Tuesday, December 8th
  - Grace days allowed
- **String Manipulation in C**
The Web in a Textbook

- Client request page, server provides, transaction done.

- A sequential server can handle this. We just need to serve one page at a time.

- This works great for simple text pages with embedded styles.
Telnet/Curl Demo

- **Telnet**
  - Interactive remote shell – like ssh without security
  - Must build HTTP request manually
    - This can be useful if you want to test response to malformed headers

```
[rjaganna@makoshark ~]# telnet www.cmu.edu 80
Trying 128.2.42.52...
Connected to WWW-CMU-PROD-VIP.ANDREW.cmu.edu (128.2.42.52).
Escape character is '^]'.
GET http://www.cmu.edu/ HTTP/1.0

HTTP/1.1 301 Moved Permanently
Date: Sat, 11 Apr 2015 06:54:39 GMT
Server: Apache/1.3.42 (Unix) mod_gzip/1.3.26.1a mod_pubcookie/3.3.4a mod_ssl/2.8.31 OpenSSL/0.9.8e-fips-rhel5
Location: http://www.cmu.edu/index.shtml
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
The document has moved <a href="http://www.cmu.edu/index.shtml">here</a>.<p>
</p><hr>
<address>
Apache/1.3.42 Server at <a href="mailto:webmaster@andrew.cmu.edu">www.cmu.edu</a> Port 80
</address>
</body></html>
```

Connection closed by foreign host.
Telnet/cURL Demo

- **cURL**
  - “URL transfer library” with a command line program
  - Builds valid HTTP requests for you!

  ```bash
  [rjaganna@makoshark ~]$ curl http://www.cmu.edu/
  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
  <HTML><HEAD>
  <TITLE>301 Moved Permanently</TITLE>
  </HEAD><BODY>
  <H1>Moved Permanently</H1>
  The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
  <HR>
  <ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cmu.edu</A> Port 80</ADDRESS>
  </BODY></HTML>
  
  ```

- Can also be used to generate HTTP proxy requests:

  ```bash
  [rjaganna@makoshark ~]$ curl --proxy lemonshark.ics.cs.cmu.edu:3092 http://www.cmu.edu/
  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
  <HTML><HEAD>
  <TITLE>301 Moved Permanently</TITLE>
  </HEAD><BODY>
  <H1>Moved Permanently</H1>
  The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
  <HR>
  <ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cmu.edu</A> Port 80</ADDRESS>
  </BODY></HTML>
  ```
How the Web Really Works

- In reality, a single HTML page today may depend on 10s or 100s of support files (images, stylesheets, scripts, etc.)

- Builds a good argument for concurrent servers
  - Just to load a single modern webpage, the client would have to wait for 10s of back-to-back request
  - I/O is likely slower than processing, so back

- Caching is simpler if done in pieces rather than whole page
  - If only part of the page changes, no need to fetch old parts again
  - Each object (image, stylesheet, script) already has a unique URL that can be used as a key
How the Web Really Works

- Excerpt from www.cmu.edu/index.html:

```html
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    ...
    <link href="homecss/cmu.css" rel="stylesheet" type="text/css"/>
    <link href="homecss/cmu-new.css" rel="stylesheet" type="text/css"/>
    <link href="homecss/cmu-new-print.css" media="print" rel="stylesheet" type="text/css"/>
    <link href="http://www.cmu.edu/RSS/stories.rss" rel="alternate" title="Carnegie Mellon Homepage Stories" type="application/rss+xml"/>
    ...
    <script language="JavaScript" src="js/dojo.js" type="text/javascript"></script>
    <script language="JavaScript" src="js/scripts.js" type="text/javascript"></script>
    <script language="javascript" src="js/jquery.js" type="text/javascript"></script>
    <script language="javascript" src="js/homepage.js" type="text/javascript"></script>
    <script language="javascript" src="js/app_ad.js" type="text/javascript"></script>
    ...
    <title>Carnegie Mellon University | CMU</title>
</head>
<body> ...
```
Sequential Proxy
Sequential Proxy

- **Note the sloped shape of when requests finish**
  - Although many requests are made at once, the proxy does not accept a new job until it finishes the current one
  - Requests are made in batches. This results from how HTML is structured as files that reference other files.

- **Compared to the concurrent example (next), this page takes a long time to load with just static content**
Concurrent Proxy
Concurrent Proxy

- Now, we see much less purple (waiting), and less time spent overall.
- Notice how multiple green (receiving) blocks overlap in time
  - Our proxy has multiple connections open to the browser to handle several tasks at once
How the Web Really Works

- A note on AJAX (and XMLHttpRequests)
  - Normally, a browser will make the initial page request then request any supporting files
  - And XMLHttpRequest is simply a request from the page once it has been loaded & the scripts are running
  - The distinction does not matter on the server side – everything is an HTTP Request
Outline

- Getting content on the web: Telnet/cURL Demo
  - How the web really works

- Networking Basics

- Proxy
  - Due Tuesday, December 8th
  - Grace days allowed

- String Manipulation in C
Sockets

- **What is a socket?**
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
  - (all Unix I/O devices, including networks, are modeled as files)

- **Clients and servers communicate with each other by reading from and writing to socket descriptors**

- The main difference between regular file I/O and socket I/O is how the application “opens” the socket descriptors
Overview of the Sockets Interface

Client
- getaddrinfo
- socket
- connect
- rio_readlineb
- rio_writen
- close

Server
- getaddrinfo
- socket
- bind
- listen
- accept
- rio_readlineb
- rio_writen
- rio_readlineb
- close

open_clientfd
- open_listenfd

Connection request
- await connection request from next client
Host and Service Conversion: `getaddrinfo`

- `getaddrinfo` is the modern way to convert string representations of host, ports, and service names to socket address structures.
  - Replaces obsolete `gethostbyname` - unsafe because it returns a pointer to a static variable

- **Advantages:**
  - Reentrant (can be safely used by threaded programs).
  - Allows us to write portable protocol-independent code (IPv4 and IPv6)
  - Given host and service, `getaddrinfo` returns result that points to a linked list of `addrinfo` structs, each pointing to socket address struct, which contains arguments for sockets APIs.

- `getnameinfo` is the inverse of `getaddrinfo`, converting a socket address to the corresponding host and service.
Sockets API

- int socket(int domain, int type, int protocol);
  - Create a file descriptor for network communication
  - used by both clients and servers
  - int sock_fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  - One socket can be used for two-way communication

- int bind(int socket, const struct sockaddr *address, socklen_t address_len);
  - Associate a socket with an IP address and port number
  - used by servers
  - struct sockaddr_in sockaddr – family, address, port
Sockets API

- int listen(int socket, int backlog);
  - socket: socket to listen on
  - used by servers
  - backlog: maximum number of waiting connections
  - err = listen(sock_fd, MAX_WAITINGgetConnections);

- int accept(int socket, struct sockaddr *address, socklen_t *address_len);
  - used by servers
  - socket: socket to listen on
  - address: pointer to sockaddr struct to hold client information after accept returns
  - return: file descriptor
Sockets API

- int connect(int socket, struct sockaddr *address, socklen_t address_len);
  - attempt to connect to the specified IP address and port described in address
  - used by clients

- int close(int fd);
  - used by both clients and servers
  - (also used for file I/O)
  - fd: socket fd to close
Sockets API

- `ssize_t read(int fd, void *buf, size_t nbyte);`
  - used by both clients and servers
  - (also used for file I/O)
  - `fd`: (socket) fd to read from
  - `buf`: buffer to read into
  - `nbytes`: buf length

- `ssize_t write(int fd, void *buf, size_t nbyte);`
  - used by both clients and servers
  - (also used for file I/O)
  - `fd`: (socket) fd to write to
  - `buf`: buffer to write
  - `nbytes`: buf length
Outline

- Getting content on the web: Telnet/cURL Demo
  - How the web really works
- Networking Basics
- Proxy
  - Due Tuesday, December 8th
  - Grace days allowed
- String Manipulation in C
Byte Ordering Reminder

- So, how are the bytes within a multi-byte word ordered in memory?
- Conventions
  - Big Endian: Sun, PPC Mac, Internet
    - Least significant byte has highest address
  - Little Endian: x86, ARM processors running Android, iOS, and Windows
    - Least significant byte has lowest address
Byte Ordering Reminder

- So, how are the bytes within a multi-byte word ordered in memory?
- Conventions
  - Big Endian: Sun, PPC Mac, Internet
    - Least significant byte has highest address
- Make sure to use correct endianness
Proxy - Functionality

- **Should work on vast majority of sites**
  - Twitch, CNN, NY Times, etc.
  - Some features of sites which require the POST operation (sending data to the website), will not work
    - Logging in to websites, sending Facebook message
  - HTTPS is not expected to work
    - Google, YouTube (and some other popular websites) now try to push users to HTTPs by default; watch out for that

- **Cache previous requests**
  - Use LRU eviction policy
  - Must allow for concurrent reads while maintaining consistency
  - Details in write up
Proxy - Functionality

■ Why a multi-threaded cache?
  ■ Sequential cache would bottleneck parallel proxy
  ■ Multiple threads can read cached content safely
    ■ Search cache for the right data and return it
    ■ Two threads can read from the same cache block
  ■ But what about writing content?
    ■ Overwrite block while another thread reading?
    ■ Two threads writing to same cache block?
Proxy - How

- Proxies are a bit special - they are a server and a client at the same time.
- They take a request from one computer (acting as the server), and make it on their behalf (as the client).
- Ultimately, the control flow of your program will look like a server, but will have to act as a client to complete the request

Start small

- Grab yourself a copy of the echo server (pg. 946) and client (pg. 947) in the book
- Also review the tiny.c basic web server code to see how to deal with HTTP headers
  - Note that tiny.c ignores these; you may not
Proxy - How

- What you end up with will resemble:

  * **Client socket address**: 128.2.194.242:51213
  * **Server socket address**: 208.216.181.15:80
  * **Proxy server socket address**: 128.2.194.34:15213
  * **Proxy client socket address**: 128.2.194.34:52943
Summary

- **Step 1: Sequential Proxy**
  - Works great for simple text pages with embedded styles

- **Step 2: Concurrent Proxy**
  - multi-threading

- **Step 3: Cache Web Objects**
  - Cache individual objects, not the whole page
  - Use an LRU eviction policy
  - Your caching system must allow for *concurrent reads* while maintaining consistency. Concurrency? Shared Resource?
Proxy – Testing & Grading

■ New: Autograder
  ▪ ./driver.sh will run the same tests as autolab:
    ▪ Ability to pull basic web pages from a server
    ▪ Handle a (concurrent) request while another request is still pending
    ▪ Fetch a web page again from your cache after the server has been stopped
  ▪ This should help answer the question “is this what my proxy is supposed to do?”
  ▪ Please don’t use this grader to definitively test your proxy; there are many things not tested here
Proxy – Testing & Grading

- **Test your proxy liberally**
  - The web is full of special cases that want to break your proxy
  - Generate a port for yourself with ./port-for-user.pl [andrewid]
  - Generate more ports for web servers and such with ./free-port.sh
  - Consider using your andrew web space (~/www) to host test files
    - You have to visit [https://www.andrew.cmu.edu/server/publish.html](https://www.andrew.cmu.edu/server/publish.html) to publish your folder to the public server

- **Create a handin file with *make handin***
  - Will create a tar file for you with the contents of your proxylab-handin folder
Outline

- Getting content on the web: Telnet/cURL Demo
  - How the web really works
- Networking Basics
- Proxy
  - Due Tuesday, December 8th
  - Grace days allowed
- String Manipulation in C
String manipulation in C

- **sscanf**: Read input in specific format

  ```c
  int sscanf(const char *str, const char *format, ...);
  ```

  Example:
  ```c
  buf = "213 is awesome"
  ```

  // Read integer and string separated by white space from buffer ‘buf’
  // into passed variables
  ```c
  ret = sscanf(buf, "%d %s %s", &course, str1, str2);
  ```

  This results in:
  ```c
  course = 213, str1 = is, str2 = awesome, ret = 3
  ```
String manipulation (cont)

- *sprintf*: Write input into buffer in specific format

  ```c
  int sprintf(char *str, const char *format, ...);
  ```

  Example:

  ```c
  buf[100];
  str = "213 is awesome"
  ```

  // Build the string in double quotes (""") using the passed arguments
  // and write to buffer ‘buf’
  ```c
  sprintf(buf, "String (%s) is of length %d", str, strlen(str));
  ```

  This results in:

  buf = String (213 is awesome) is of length 14
String manipulation (cont)

Other useful string manipulation functions:

- strcmp, strncmp, strncasecmp
- strstr
- strlen
- strcpy, strncpy
Aside: Setting up Firefox to use a proxy

- You may use any browser, but we’ll be grading with Firefox
- Preferences > Advanced > Network > Settings… (under Connection)
- Check “Use this proxy for all protocols” or your proxy will appear to work for HTTPS traffic.
Acknowledgements

- Slides derived from recitation slides of last 2 years by
  - Shiva
  - Hartaj Singh Dugal
  - Ian Hartwig
  - Rohith Jagannathan
Questions?