

# Proxy Recitation

Minsu Kang

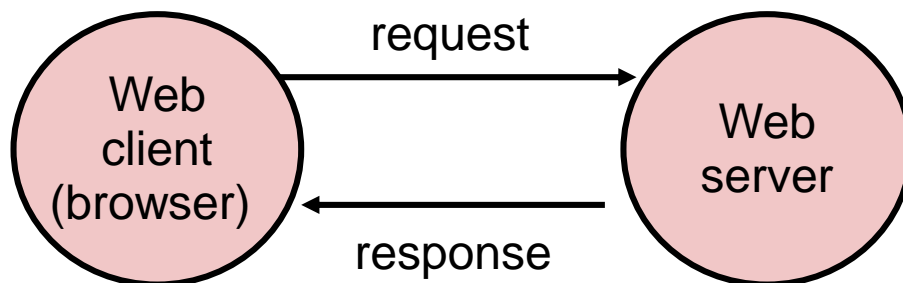
Recitation 7: May 22, 2019

# Outline

- **Getting content on the web: Telnet/cURL Demo**
  - How the web really works
- Networking Basics
- Proxy
  - Due Wednesday, June 5th
- 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_public/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>
<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!

```
[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:

```
[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>
```

# Web-browser Demo

## ■ Chrome Developer Console

- Type domain name (ip address) on search bar, press F12 key and see that what codes have been sent from web server.
- [Elements tab]

The screenshot shows a web browser displaying the UNIST website. The browser's address bar shows the URL `https://www.unist.ac.kr`. The website features a navigation bar with links like "About UNIST", "Admissions", "Academics", "Research", and "Campus Life". The main content area includes sections for "Community Contribution", "EVENT", "UNIST NEWS" (highlighting "THE Young University Rankings 2018: UNIST Ranked No. 5 Worldwide"), "Admission" (for Undergraduate and Graduate), "WORLD UNIVERSITY RANKINGS", "UNIST Industry-University Convergence Campus", and "CAMPUS MAP".

The Chrome Developer Console is open, showing the "Elements" tab. The HTML structure is visible, starting with the `<html>` tag, followed by `<head>` and `<body>` tags. The `<head>` section contains various meta tags, links to external resources (like fonts and stylesheets), and a script tag. The `<body>` tag is the root of the page content.

Below the Elements tab, the "Highlights from the Chrome 74 update" section is visible, providing information about new features and updates.

# Web-browser Demo

## ■ Chrome Developer Console

- Type domain name (ip address) on search bar, press F12 key and see that what codes have been sent from web server.
- [Network tab]

The screenshot displays the UNIST (Ulsan National Institute of Science and Technology) website in a web browser. The URL bar shows `https://www.unist.ac.kr`. The website features a navigation menu with links like 'About UNIST', 'Admissions', 'Academics', 'Research', and 'Campus Life'. A search bar is located at the top right. The main content area includes a 'CAMPUS MAP', 'Community Contribution', 'EVENT', 'Admission' (Undergraduate and Graduate), 'UNIST Identity', and 'THE Young University Rankings 2018: UNIST Ranked No. 5 Worldwide'. The Chrome Developer Console is open on the right side, showing the Network tab. The console displays a list of resources loaded from the website, including CSS files, JavaScript files, and images. The table below summarizes the resources shown in the console:

Name	Status	Type	Initiator	Size	Time	Waterfall
www.unist.ac.kr	200	document	Other	47.3 KB	743 ms	
custom.css?v=201905203	200	stylesheet	(index)	(from ...)	5 ms	
responsive.css	200	stylesheet	(index)	(from ...)	5 ms	
font-awesome.min.css	200	stylesheet	(index)	(from ...)	3 ms	
font-awesome.css	200	stylesheet	(index)	(from ...)	3 ms	
metabrain.css	200	stylesheet	(index)	(from ...)	1 ms	
m_css.css	200	stylesheet	(index)	(from ...)	2 ms	
user_ahw.css	200	stylesheet	(index)	(from ...)	2 ms	
classiclight.css?ver=0.9.93	200	stylesheet	(index)	(from ...)	2 ms	
styles.css?ver=4.0.3	200	stylesheet	(index)	(from ...)	3 ms	
dashicons.min.css?ver=4.7.2	200	stylesheet	(index)	(from ...)	4 ms	
frontend.css?ver=1.2.7	200	stylesheet	(index)	(from ...)	4 ms	
css?family=Lato%3A300%2...	200	stylesheet	(index)	(from ...)	4 ms	
genericons.css?ver=3.0.3	200	stylesheet	(index)	(from ...)	4 ms	
style.css?ver=4.7.2	200	stylesheet	(index)	(from ...)	5 ms	
jquery.js?ver=1.12.4	200	script	(index)	(from ...)	5 ms	
jquery-migrate.min.js?ver=1...	200	script	(index)	(from ...)	4 ms	
jquery.archivesCW.min.js?ve...	200	script	(index)	(from ...)	4 ms	
magnific-popup.css?ver=4...	200	stylesheet	(index)	(from ...)	4 ms	
concentration_eng.png	200	png	(index)	(from ...)	2 ms	
event.png	200	png	(index)	(from ...)	1 ms	
jquery.form.min.js?ver=3.51...	200	script	(index)	(from ...)	1 ms	
scripts.js?ver=4.0.3	200	script	(index)	(from ...)	1 ms	
comment-reply.min.js?ver=...	200	script	(index)	(from ...)	1 ms	
imagesloaded.min.js?ver=3...	200	script	(index)	(from ...)	1 ms	
css?family=Open+Sans:300...	200	stylesheet	(index)	(from ...)	1 ms	
css?family=Source+Sans+P...	200	stylesheet	(index)	(from ...)	1 ms	
metabrain.css	200	stylesheet	(index)	(from ...)	1 ms	

The Chrome Developer Console also shows the Console tab with a message about the Chrome 74 update and the Lighthouse v4 audit panel. The Network tab is highlighted with a red box.

# 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 <https://www.unist.ac.kr/>:

The screenshot shows the UNIST website with the browser developer tools open. The website features a header with the UNIST logo, navigation links (About UNIST, Admissions, Academics, Research, Campus Life), and a search bar. The main content area includes a grid of tiles for Community Contribution, Event, and Admissions. The Admissions tile is highlighted, showing the 'Undergraduate' and 'Graduate' links. The developer tools show the HTML and CSS for the 'Admission' section, including the 'UNIST Admission' title and the 'Undergraduate' and 'Graduate' links.

**HTML Excerpt:**

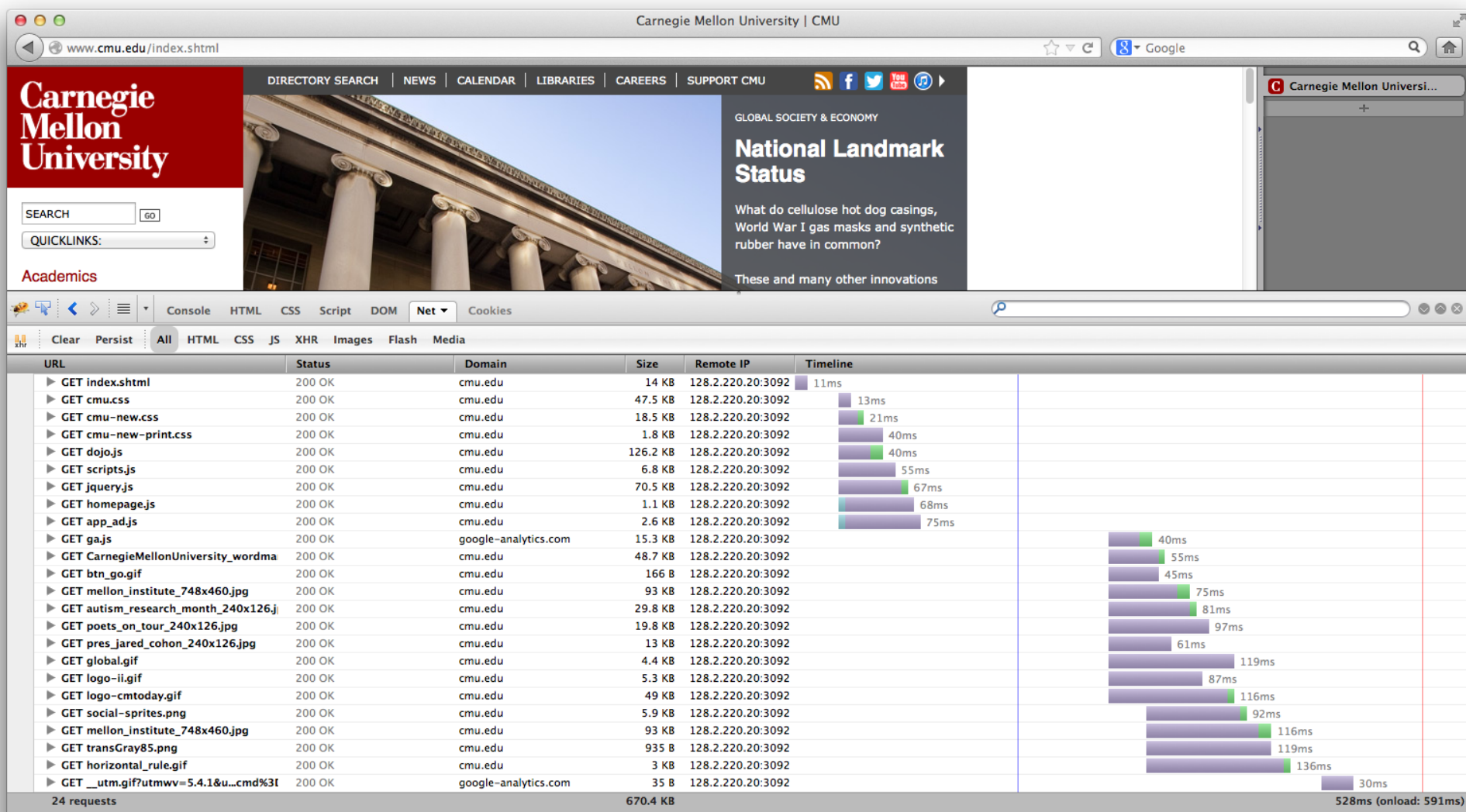
```
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif-->
<!--[if (IE 7) & (IE 8)]><!-->
<html lang="en-US">
<!--[endif-->
<!--[if IE 8]>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=2">
<meta name="format-detection" content="telephone=no">
<title>UNIST</title>
<link rel="profile" href="http://ogp.org/xfn/11">
<link rel="shortcut icon" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/images/favicon.ico">
<link rel="pingback" href="https://www.unist.ac.kr/xmlrpc.php">
<link rel="stylesheet" type="text/css" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/print.css" media="print">
<link rel="stylesheet" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/custom.css?u=20190929">
<link rel="stylesheet" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/responsive.css">
<link rel="stylesheet" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/font/css/font-awesome.min.css">
<link rel="stylesheet" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/font/css/font-awesome.min.css">
<link rel="stylesheet" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/metafont.css">
<link rel="stylesheet" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/m.css">
<link rel="stylesheet" href="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/css/user_ahw.css">
<!--[if !IE 9]>
<script src="https://www.unist.ac.kr/wp-content/themes/twentyfourteen/js/html5.js"></script>
<![endif-->
<!-- All in One SEO Pack 2.3.11.4 by Michael Torbert of Semper Fi Web Design[1,-1] ->
<!--
<link rel="author" href="http://unist">
<link rel="publisher" href="http://unist">
<meta name="description" content="To be Ranked within the Top 10 Science and Technology University by 2030">
<meta name="keywords" content="UNIST, unist">
<link rel="canonical" href="https://www.unist.ac.kr/">
<!-- /all in one seo pack -->
<link rel="dns-prefetch" href="//fonts.googleapis.com">
<link rel="dns-prefetch" href="//s.w.org">
<link rel="alternate" type="application/rss+xml" title="UNIST » Feed" href="https://www.unist.ac.kr/feed/">
</body>
```

**CSS Excerpt:**

```
.show .cls
element.st
yle {
}
(index):88
body.custo
m
background
{
background-
color
#fff
}
style.css-
body {
width:
100%;
}
style.css-
body {
font-
family:
'Sour
Sans
Pro',
sans-
serif
!impor
tant;
color:
#222;
}
style.css-
body {
background:
#fff;
margin:
0;
padding:
0;
}
style.css-
body,
button,
input,
select,

```

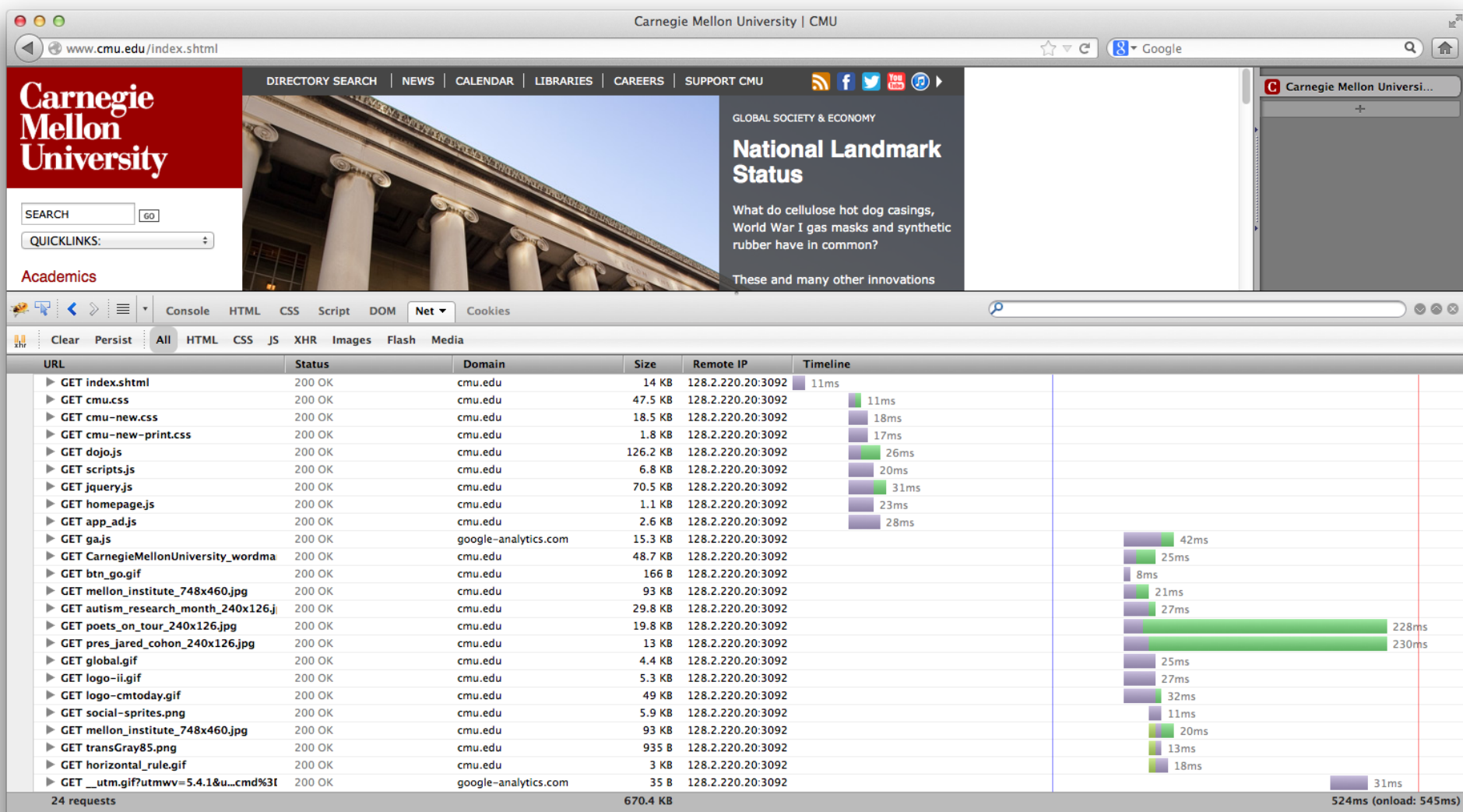
# Sequential Proxy



# 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

\*AJAX: [https://en.wikipedia.org/wiki/Ajax\\_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))

# Outline

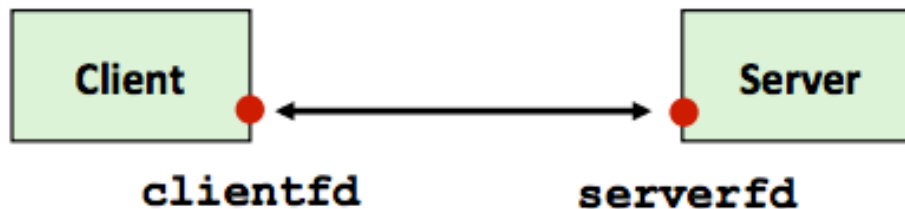
- Getting content on the web: Telnet/cURL Demo
  - How the web really works
- **Networking Basics**
- Proxy
  - Due Wednesday, June 5th
- String Manipulation in C

# 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

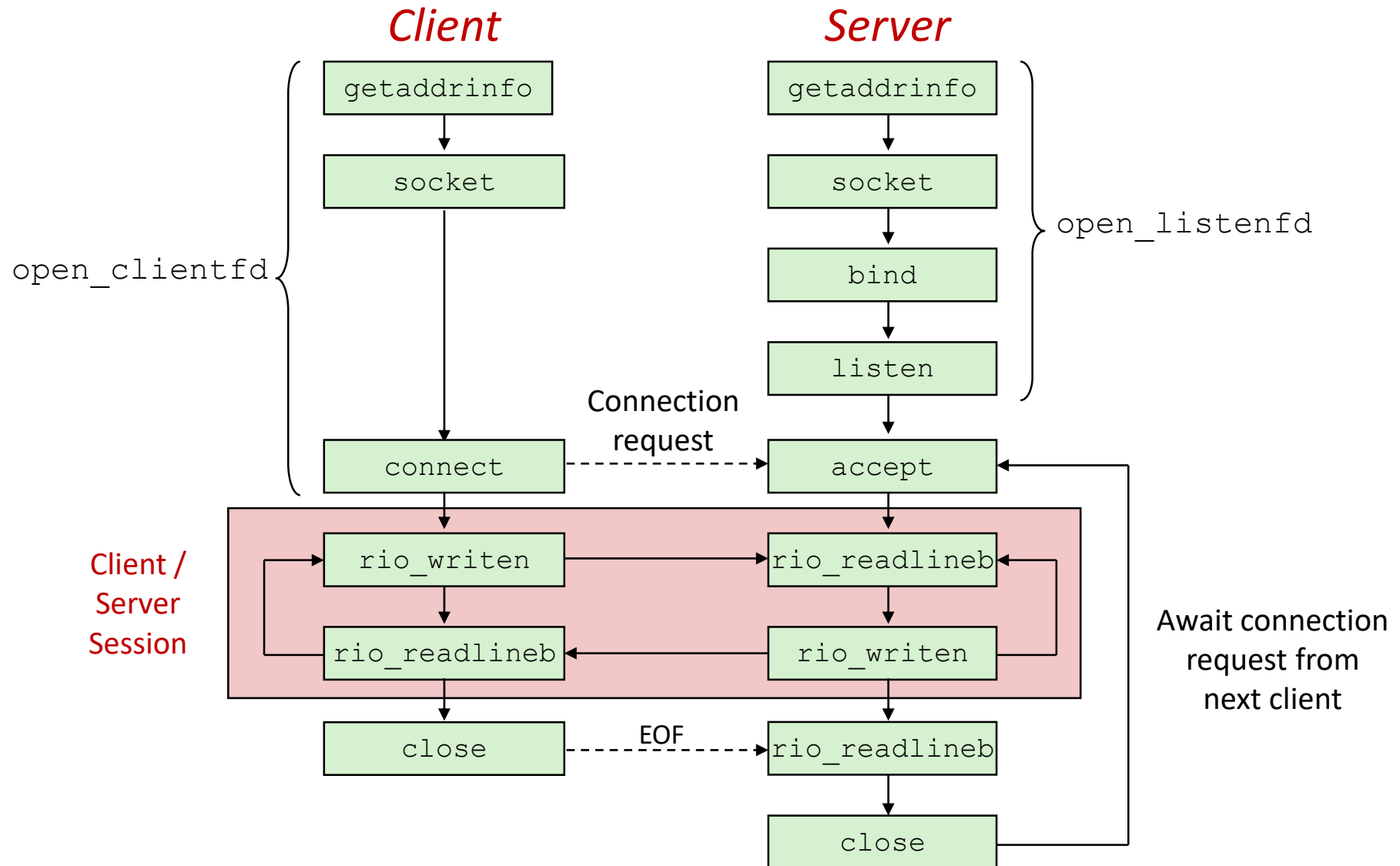


## ■ The main difference between regular file I/O and socket I/O is how the application “opens” the socket descriptors

\*file descriptor: <https://stackoverflow.com/questions/30682057/socket-programming-what-is-fd-and-sd>  
[https://en.wikipedia.org/wiki/File\\_descriptor#Sockets](https://en.wikipedia.org/wiki/File_descriptor#Sockets)



# Overview of the Sockets Interface



# 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_WAITING_CONNECTIONS);`

## ■ **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 Wednesday, June 5th
- 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 for concurrency?
  - 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?

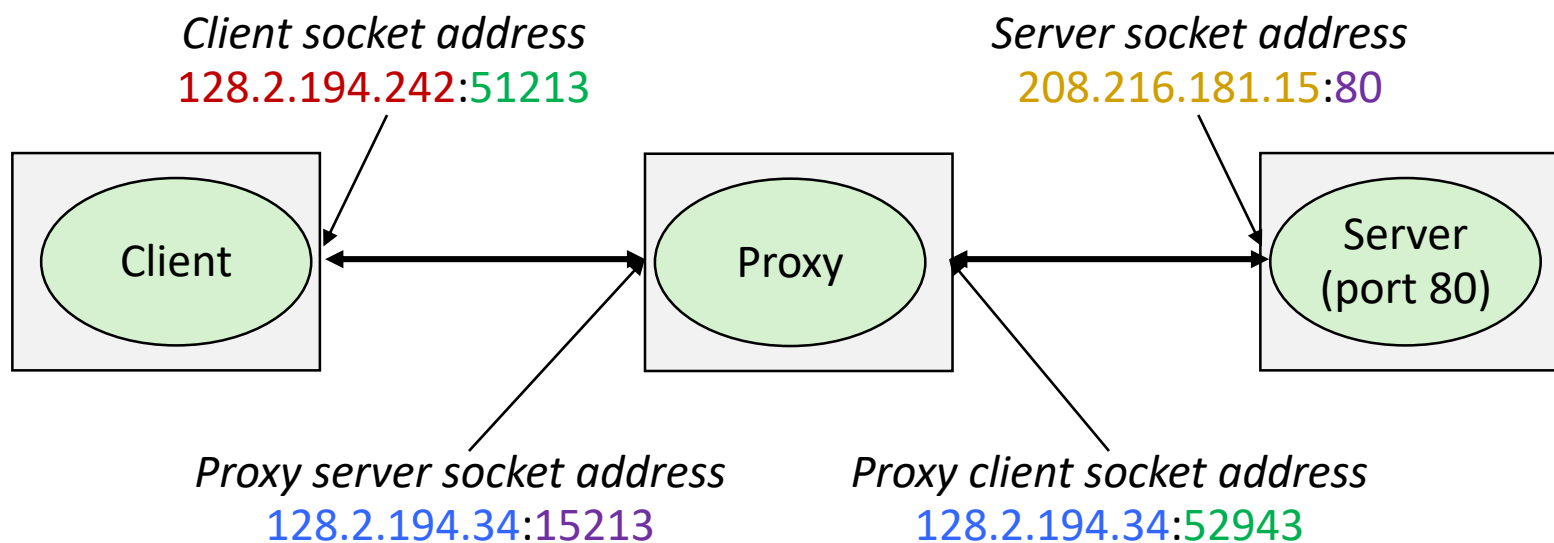
**\*multi-threaded:** if you don't know about multi-threading, please refer slide 38

# 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:



# 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

# Outline

- Getting content on the web: Telnet/cURL Demo
  - How the web really works
- Networking Basics
- Proxy
  - Due Wednesday, June 5th
- **String Manipulation in C**



# String manipulation in C

## ■ sscanf: Read input in specific format

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

Example:

```
buf = "213 is awesome"
```

```
// Read integer and string separated by white space from buffer 'buf'
```

```
// into passed variables
```

```
ret = sscanf(buf, "%d %s %s", &course, str1, str2);
```

This results in:

course = 213, str1 = is, str2 = awesome, ret = 3

# String manipulation (cont)

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

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

Example:

*buf[100];*

*str = "213 is awesome"*

*// Build the string in double quotes ("" ) using the passed arguments*

*// and write to buffer 'buf'*

*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`

# Warnings!

## ■ Please read the instruction carefully!

- Please “README.md” and “proxylab.pdf”  
 , then you will get to know how to do.

## ■ You may utilize pre-implemented functions in csapp.c and cache.c file.

- Utilize them as much as you can!
- Or you can implement your own code  
 , but I don't recommend it.

## ■ Do proxy lab in a linux machine

## ■ Got in a serious trouble?

- Please email me ([mins@unitst.ac.kr](mailto:mins@unitst.ac.kr))  
 (only for a serious trouble)

```
[mins@uni06 proxylab]$ ./driver.sh
*** Basic ***
Starting tiny on 19448
Starting proxy on 17315
1: home.html
  Fetching ./tiny/home.html into ./proxy using the proxy
  Fetching ./tiny/home.html into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
2: csapp.c
  Fetching ./tiny/csapp.c into ./proxy using the proxy
  Fetching ./tiny/csapp.c into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
3: tiny.c
  Fetching ./tiny/tiny.c into ./proxy using the proxy
  Fetching ./tiny/tiny.c into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
4: godzilla.jpg
  Fetching ./tiny/godzilla.jpg into ./proxy using the proxy
  Fetching ./tiny/godzilla.jpg into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
5: tiny
  Fetching ./tiny/tiny into ./proxy using the proxy
  Fetching ./tiny/tiny into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
Killing tiny and proxy
basicScore: 40/40

*** Concurrency ***
Starting tiny on port 16068
Starting proxy on port 22691
Starting the blocking NOP server on port 29653
Trying to fetch a file from the blocking nop-server
Fetching ./tiny/home.html into ./noproxy directly from Tiny
Fetching ./tiny/home.html into ./proxy using the proxy
Checking whether the proxy fetch succeeded
Success: Was able to fetch tiny/home.html from the proxy.
Killing tiny, proxy, and nop-server
concurrencyScore: 15/15

*** Cache ***
Starting tiny on port 12323
Starting proxy on port 14826
Fetching ./tiny/tiny.c into ./proxy using the proxy
Fetching ./tiny/home.html into ./proxy using the proxy
Fetching ./tiny/csapp.c into ./proxy using the proxy
Killing tiny
Fetching a cached copy of ./tiny/home.html into ./noproxy
Success: Was able to fetch tiny/home.html from the cache.
Killing proxy
cacheScore: 15/15

totalScore: 70/70
```

<Results of proxy lab>

# Questions?

# Some Useful References

## ■ Regarding Proxy Lab

### ■ Network Programming in C

- Article

<https://www.geeksforgeeks.org/socket-programming-cc/>

- Video Lecture (Socket Programming Tutorial using C)

<https://www.youtube.com/watch?v=LtXEMwSG5-8>

### ■ Multithreading in C

- Article

<https://www.geeksforgeeks.org/multithreading-c-2/>

- Video Lecture (Threading Basics in C)

<https://www.youtube.com/watch?v=nVESQQg-Oiw>

# Some Useful References

## ■ Other Materials

- For completing proxy lab, you don't need to know deeply about concepts which were represented in the following links, but I think learning them will be helpful to you.
- **Process vs Thread**
  - <https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/>
- **Multiprocessing vs Multithreading**
  - <https://www.youtube.com/watch?v=oIN488Ldg9k>
- **OSI 7 Layer**
  - [https://www.youtube.com/watch?v=vv4y\\_uOneC0](https://www.youtube.com/watch?v=vv4y_uOneC0)
- **Differences between TCP and UDP**
  - <https://www.howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp/>