Concurrency & Proxy Lab

Recitation 13: November 28, 2017

Part A

- **Visualizing Concurrency: Progress Graphs**

Assembly Code for Counter Loop

C code for counter loop in thread i

```
for (i = 0; i < niters; i++)
    cnt++;
```

Asm code for thread i

```
    movq  (%rdi), %rcx
    testq %rcx,%rcx
    jle   .L2
    movl  $0, %eax
.L3:
    movq  cnt(%rip),%rdx
    addq  $1, %rdx
    movq  %rdx, cnt(%rip)
    addq  $1, %rax
    cmpq  %rcx, %rax
    jne   .L3
.L2:
```

H_i : Head

L_i : Load `cnt`

U_i : Update `cnt`

S_i : Store `cnt`

T_i : Tail

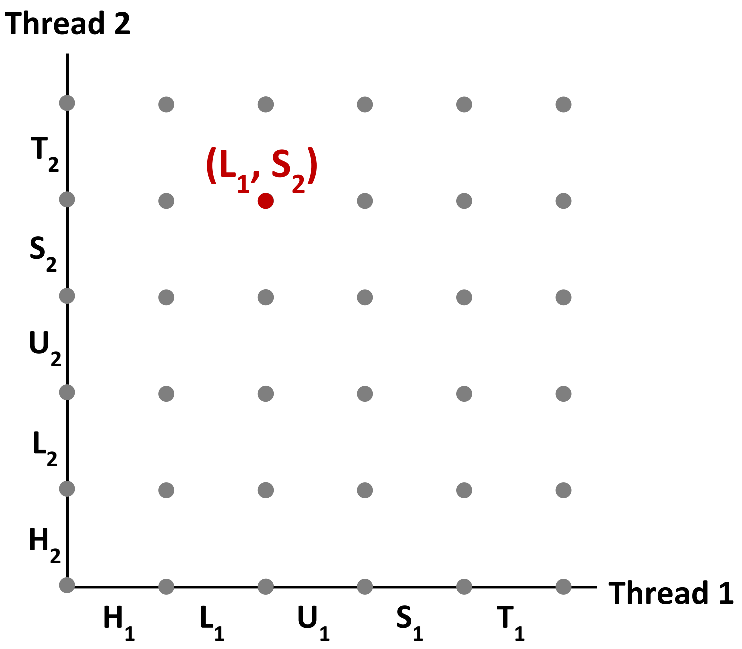
Visualizing Concurrency: Progress Graphs

- Recall this ordering from yesterday's lecture slide 10
  - Incorrect ordering: two threads increment the counter, but the result is 1 instead of 2

i (thread)	$instr_i$	$\%rdx_1$	$\%rdx_2$	cnt	
1	H_1	-	-	0	
1	L_1	0	-	0	
1	U_1	1	-	0	
2	H_2	-	-	0	
2	L_2	-	0	0	
1	S_1	1	-	1	
1	T_1	1	-	1	
2	U_2	-	1	1	
2	S_2	-	1	1	
2	T_2	-	1	1	*Oops!*

- We can analyze the behavior using a ***progress graph***

Progress Graphs



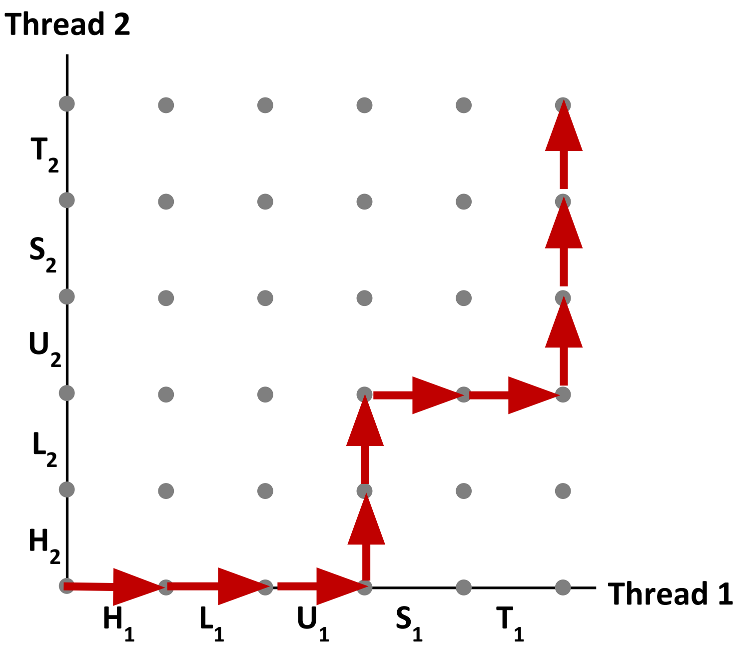
A *progress graph* depicts the discrete *execution state space* of concurrent threads.

Each axis corresponds to the sequential order of instructions in a thread.

Each point corresponds to a possible *execution state* $(Inst_1, Inst_2)$.

E.g., (L_1, S_2) denotes state where thread 1 has completed L_1 and thread 2 has completed S_2.

Trajectories in Progress Graphs

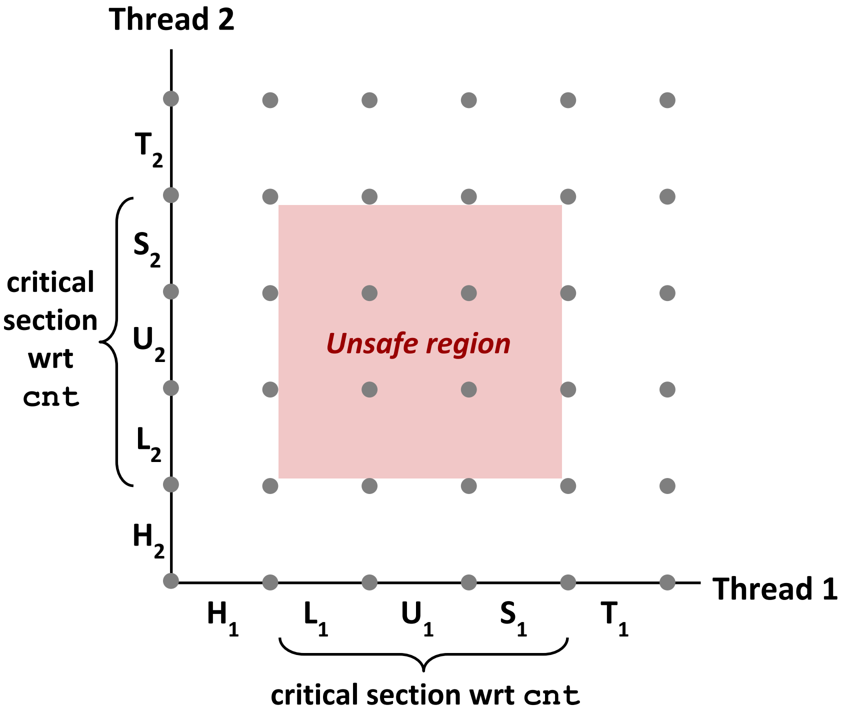


A *trajectory* is a sequence of legal state transitions that describes one possible concurrent execution of the threads.

Example:

H1, L1, U1, H2, L2, S1, T1, U2, S2, T2

Critical Sections and Unsafe Regions

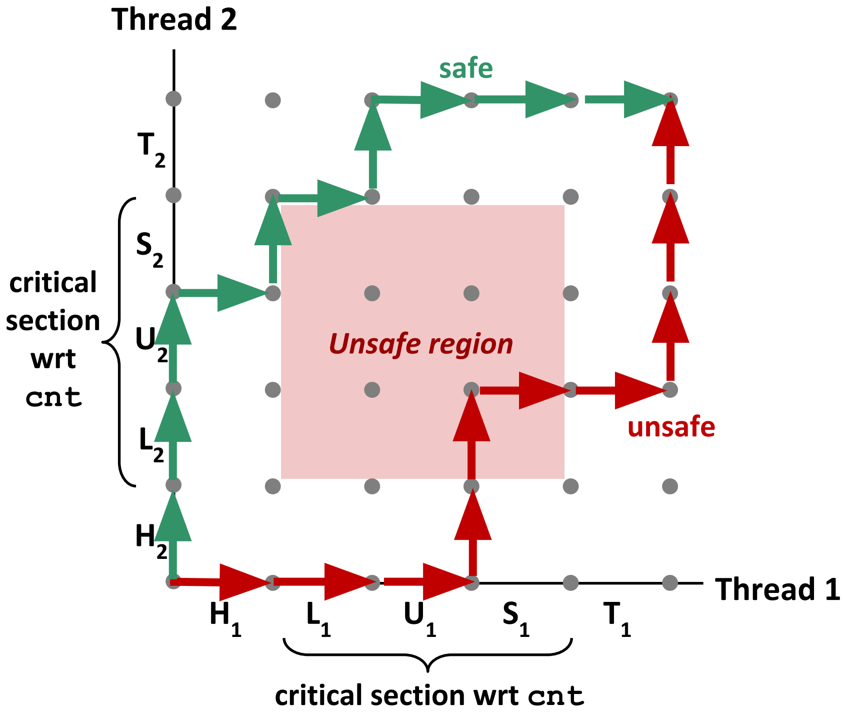


L, U, and S form a *critical section* with respect to the shared variable `cnt`

Instructions in critical sections (wrt some shared variable) should not be interleaved

Sets of states where such interleaving occurs form *unsafe regions*

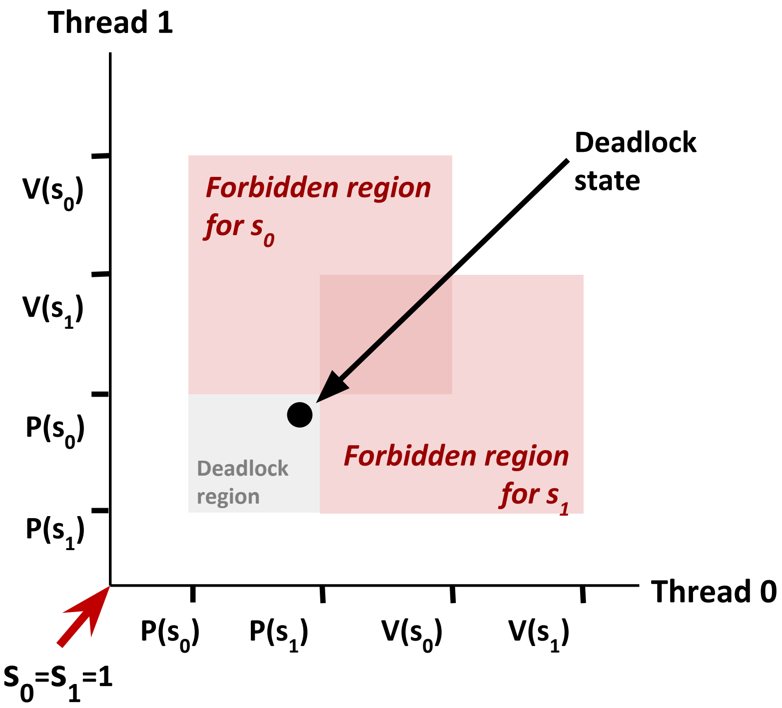
Trajectory safeness and correctness



Def: A trajectory is *safe* iff it does not enter any unsafe region

Claim: A trajectory is correct (wrt `cnt`) iff it is safe

Deadlock Visualized in Progress Graph



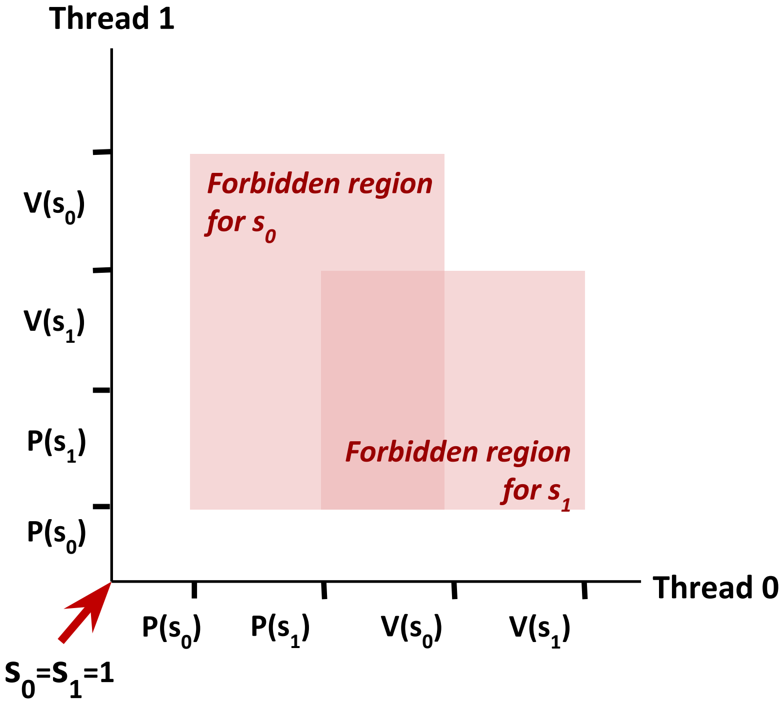
Locking introduces the potential for *deadlock:* waiting for a condition that will never be true

Any trajectory that enters the *deadlock region* will eventually reach the *deadlock state,* waiting for either s_0 or s_1 to become nonzero

Other trajectories luck out and skirt the deadlock region

Unfortunate fact: deadlock is often nondeterministic (race)

Avoided Deadlock in Progress Graph



No way for trajectory to get stuck

Processes acquire locks in same order

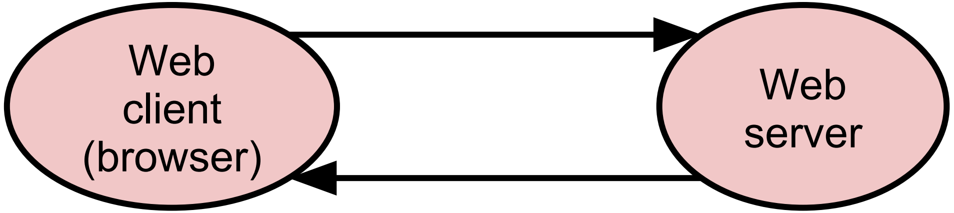
Order in which locks released immaterial

Part B

- **Getting content on the web: Telnet/cURL**
  - How the web really works
- **Networking Basics**
- **Echo Client & Server Demo**
- **Proxy**
  - Due Tuesday, December 12th
  - 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

- **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>
<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

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

```
[prodney@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>
```

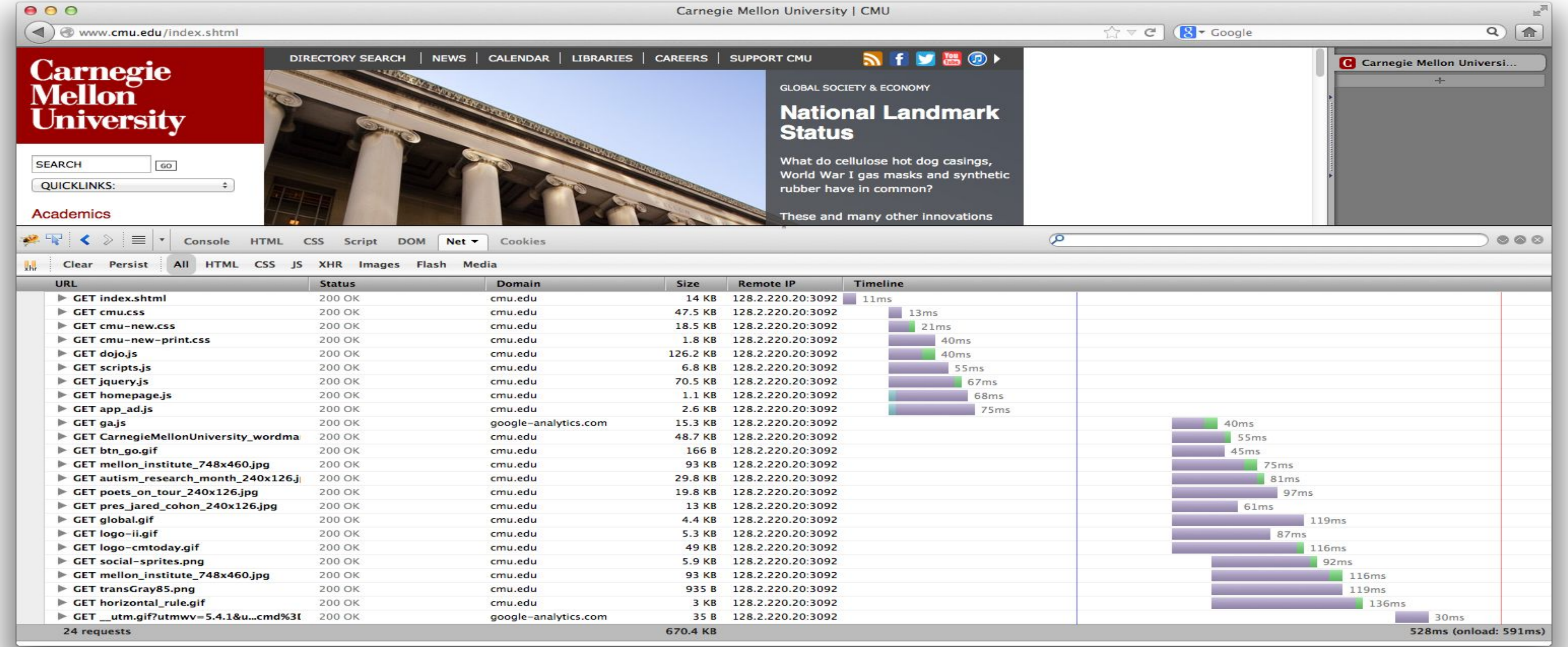
  - **Can also be used to generate HTTP proxy requests:**

```
[prodney@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

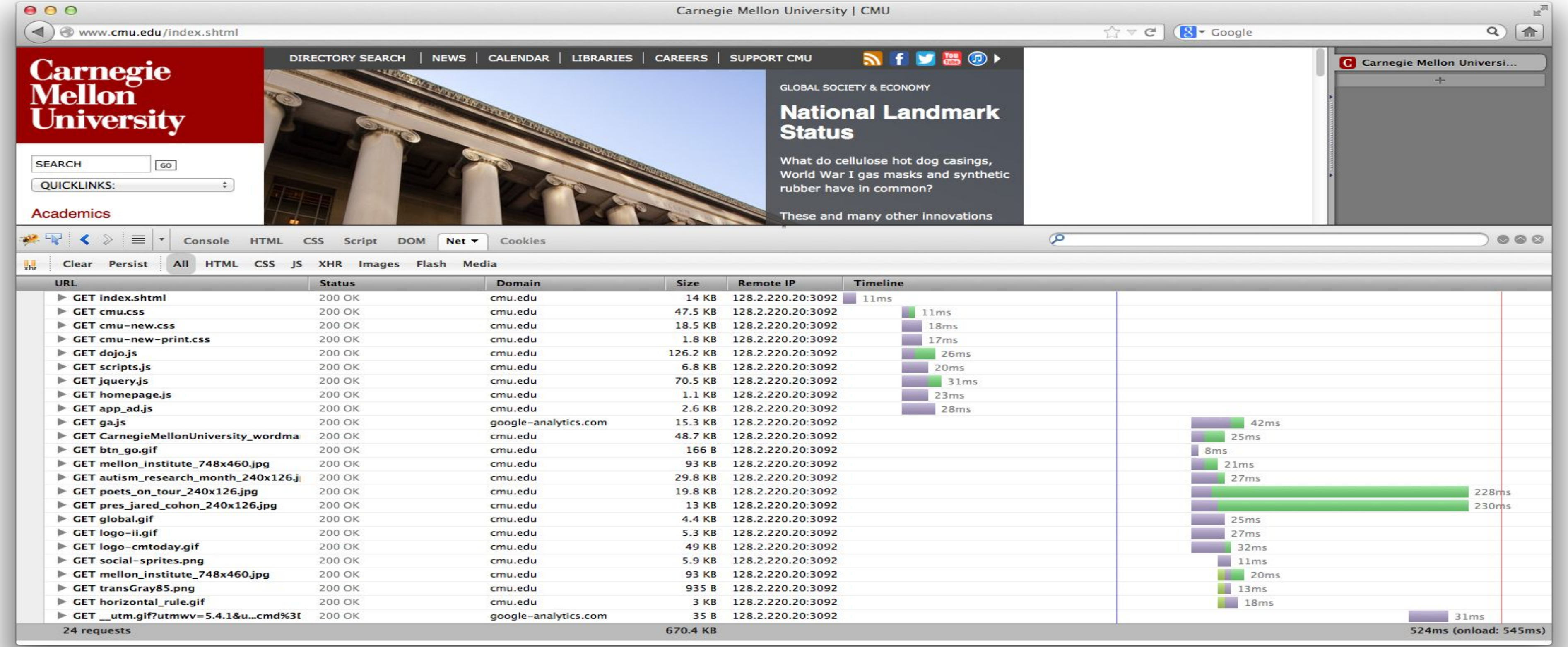
Sequential Proxy



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

Part B

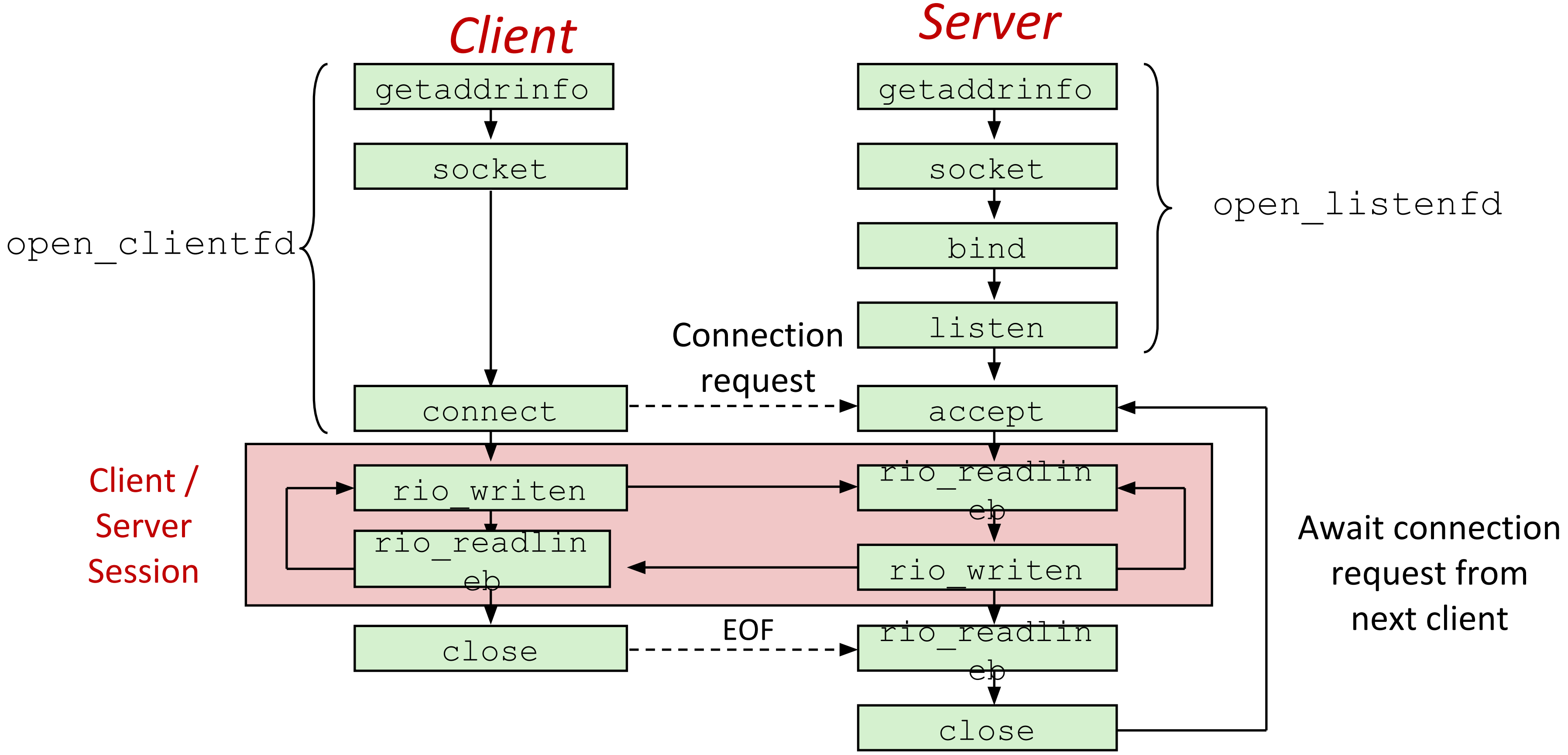
- Getting content on the web: Telnet/cURL Demo
  - How the web really works
- Networking Basics
- Echo Client & Server Demo
- Proxy
  - Due Tuesday, December 12^{th}
  - Grace days allowed
- String Manipulation in C

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



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

Part B

- Getting content on the web: Telnet/cURL
  - How the web really works
- Networking Basics
- Echo Client & Server Demo
- Proxy
  - Due Tuesday, December 12th
  - 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

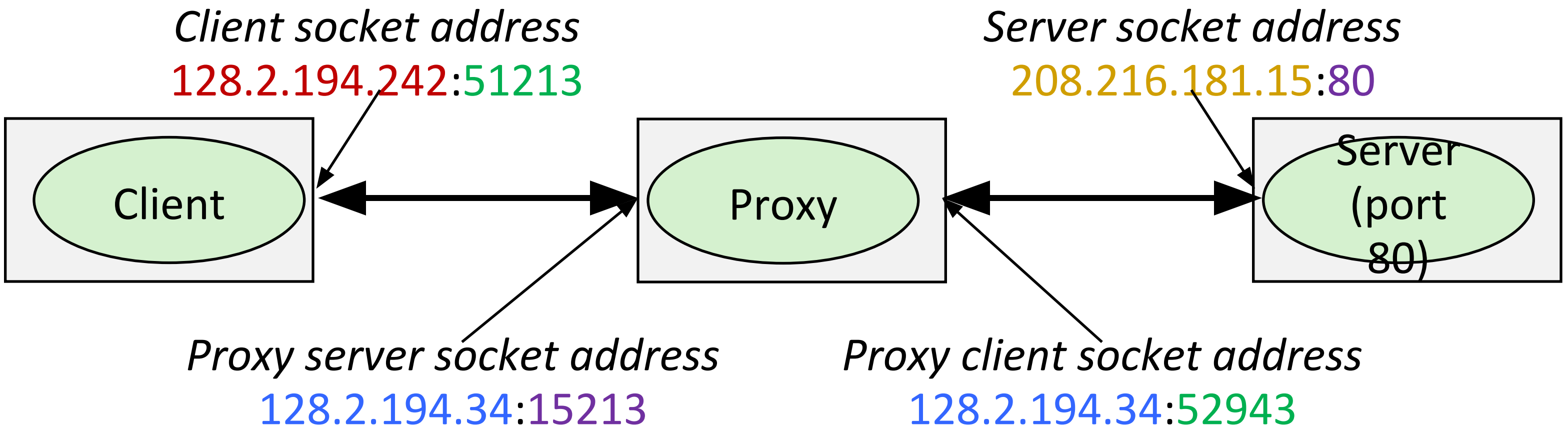
- **So, how are the bytes within a multi-byte word ordered in memory?**
- **Conventions**
    - Big Endian: Sun, PPC Mac, **Internet**
        - Least significant byte has highest address
- **Make sure to use correct endianness**

Proxy - 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:**



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 into 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?

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

- **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 (think small images, large images, videos, etc.)
  - Generate a port for yourself with ./port-for-user.pl [andrewid]
  - Generate more ports for web servers and such with ./free-port.sh
- **Create a handin file with *make handin***
  - Will create a tar file for you with the contents of your proxylab-handin folder

Part B

- Getting content on the web: Telnet/cURL
  - How the web really works
- Networking Basics
- Echo Client & Server Demo
- Proxy
  - Due Tuesday, December 12th
  - Grace days allowed
- **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)

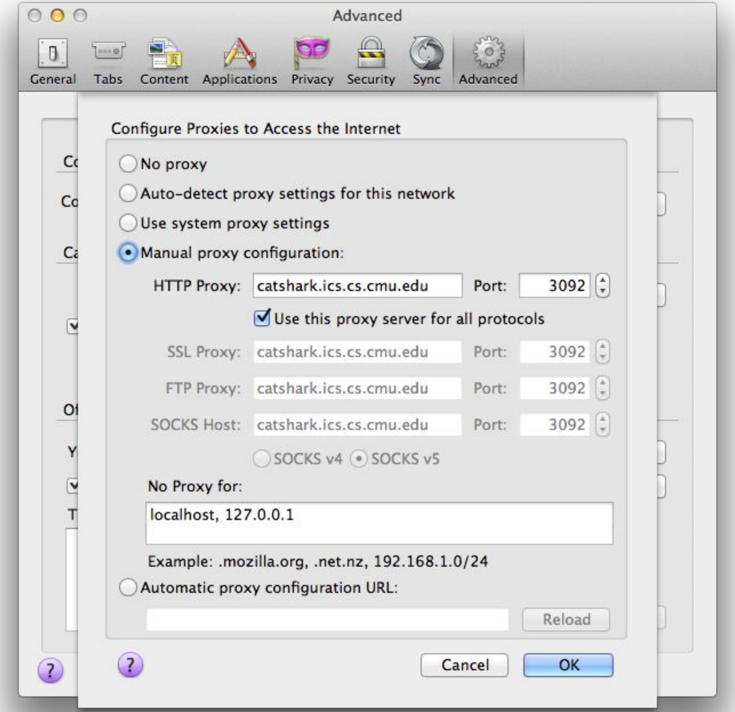
Other useful string manipulation functions:

- strcmp, strncmp, strncasecmp
- strstr
- strlen
- strcpy, strncpy

String Manipulation (cont)

- **Beware: String operations will NOT work properly with binary data**
  - E.g. images, videos, etc
  - Think about the null terminator string operations check for
  - Remember this when caching data objects
- **Solution: use memcpy instead**
  - **void *memcpy(void ***_dest_**, const void ***_src_**, size_t** _n_**);**

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.

Questions?