MP6 – Web Proxy overview

CS241 Sp10

What is the MP about?

- Networking
  - Multi-threaded web server
  - Web client
  - The server should respond to client requests with local pages or by fetching remote internet pages
- HTTP 1.1
  - Reduced set of functions
  - Interpret the requests
  - Send error messages
  - Make requests to another server
- Resources
  - You can use code snippets from class or the Beej’s guide
  - If you do, put a reference in the comments of your code

Task I - Listen

- **`argv[1]`** is the port you should listen to for incoming connections
  - Choose a port between 1024 and 32000
  - On the same host only one proxy can listen on a single port, choose random port# to avoid overlaps
- Steps:
  - Create a socket
    - set option SO_REUSEADDR to avoid bind failures
  - Bind to the host address
  - Listen
  - Accept

Task I - Listen - Testing

- Run your program
  - `./proxy <port#>`
- Open a telnet connection on the same host
  - `telnet localhost <port#>`
- Your server was blocked on accept() before the client connects, accept should return a file descriptor for the new connection

Task 2 – Threads

- Your proxy must handle multiple connections at the same time:
  - Requests from different clients
  - Concurrent requests from the same browser
- **`accept()`** must be called in a loop
  - Every incoming connection spawns a new thread that handles it
  - Thread, not processes!
  - The main thread keeps waiting for new connections and spawns new threads as they come

Task 2 – Threads - Testing

- Have the program print a message on stdout when a new thread is started

- Run your program
  - `./proxy <port#>`

- Open a telnet connection on the same host
  - `telnet localhost <port#>`
- Open another telnet connection on the same host
  - `telnet localhost <port#>`

- If your server is correctly spawning threads the second telnet should create a new connection and your proxy should print two messages on stdout

Task 2 – Threads - Testing

- Have the program print a message on stdout when a new thread is running

- Run your program
  - `./proxy <port#>`

You can now READ and WRITE as you would do with a pipe. Time to behave as a webserver now!

- If your server is correctly spawning threads the second telnet should create a new connection and your proxy should print two messages on stdout

Task 3 – HTTP 1.1 Request

- HTTP 1.1 request sent by a browser for **`index.html`**

```
GET /index.html HTTP/1.1
Host: localhost:1234
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US)
AppleWebKit/532.5 (KHTML, like Gecko) Chrome/4.0.249.43
Safari/532.5
Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,text/pl
ain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

```

Task 3 – HTTP 1.1 Request

- HTTP 1.1 request sent by a browser for **`index.html`**

```
GET /index.html HTTP/1.1
Host: localhost:1234
```

FILENAME and PROTOCOL

`GET /index.html HTTP/1.1`

Specifies what file the client is asking for

Task 3 – HTTP 1.1 Request

- HTTP 1.1 request sent by a browser for **`index.html`**

```
GET /index.html HTTP/1.1
Host: localhost:1234
```

Host: (REQUIRED)

`Host: localhost:1234`

HTTP 1.1 supports multiple hostnames on the same IP. This header is how it is done. Every request must include this field

Task 3 – HTTP 1.1 Request

Listen · Threads · Request

- HTTP 1.1 request sent by a browser for **`index.html`**

```
GET /index.html HTTP/1.1
Host: localhost:1234
```

Persistent connection

`Connection: keep-alive`

If "keep-alive" the browser will be sending more requests on the same socket. If "close" the server is expected to close the socket at the end of the current response.

Task 3 – HTTP 1.1 Request

- To extract the filename from the request:

```
getFileNameFromHTTPRequest(void *vptrRequest, size_t length)
```

- Possible return values:
  - **`NULL`** → The request is not valid
  - **`/`** → The default webpage (/index.html)
  - **`/path_to_filename/filename`** → The filename the client is requesting
  - **`/proxy/<hostname>/filepath`** → The request is for an external website

Task 3 – HTTP 1.1 Request

- To extract the filename from the request:

```
getFileNameFromHTTPRequest(void *vptrRequest, size_t length)
```

Webserver ROOT directory:

All requests for a local file must be solved by looking for the files in the ./pages/ folder and its subfolders.

- **/proxy/<hostname>/filepath** > The request is for an external website

Task 3 – HTTP 1.1 Request – Testing

- How do I know if I correctly receive the requests?
  - Have your thread printing on screen everything it reads
  - Launch your proxy
  - Open a web browser and open the address:
    - http://<hostname>:<port#>
  - Your server should print something that looks like the example in the previous slides

- How do I know the hostname of the machine I’m running my proxy on?
  - Simply run “hostname” from a terminal

Task 4 – HTTP 1.1 Response

- If the request can be satisfied the response starts with:

```
GET /index.html HTTP/1.1

HTTP/1.1 200 OK
Server: CS/241
MIME-version: 1.0
Content-type: text/html
Content-Length: 1147
```

  - Load the file in a char*, have its length ready too

```
getResponseString(char *sContentType, void *vptrContent,
    size_t iContentLength)
```

```
text/html
text/css
image/png
image/jpeg
```

Task 4 – HTTP 1.1 Response

- Example of a response for a JPEG image

```
GET images/lena.jpg

HTTP/1.1 200 OK
Server: CS/241
MIME-version: 1.0
Content-type: image/jpeg
Content-Length: 84360
```

Task 4 – HTTP 1.1 Response

- If the file does not exist and the path does not start with /proxy/:

`getFileNotFoundResponseString()`

Returns the complete response below (in white)

```
GET /notafile.html HTTP/1.1

HTTP/1.1 404 Not Found
Server: CS/241
Connection: close
Content-Length: 300

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html
xmlns="http://www.w3.org/1999/xhtml"><head><title>HTTP/404
File Not Found</title></head><body><h1>Not Found</h1><div>Your
requested file was not found on the
server.</div></body></html>
```

Task 4 – HTTP 1.1 Response

Listen
Threads
Request
Prepare the response

- If the request is not a GET request:

`getNotImplementedResponseString()`

Returns the complete response below (in white)

```
NOTAREQUEST

HTTP/1.1 501 Not Implemented
Server: CS/241
Connection: close
Content-Length: 315

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html
xmlns="http://www.w3.org/1999/xhtml"><head><title>HTTP/501 Not
Implemented</title></head><body><h1>Not
Implemented</h1><div>Your requested cannot be undersood by the
server. Sorry.</div></body></html>Connection closed by foreign
host.
```

Task 5 – Answer the Request

- We have a response ready in a **HTTPResponse struct:**

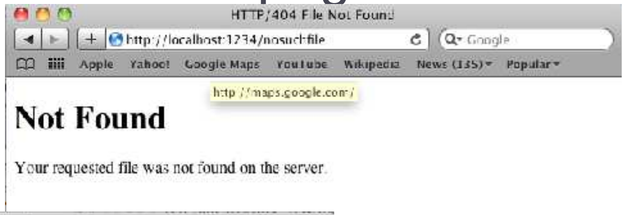
```
typedef struct __HTTPResponse{
    void *vptrResponse;
    size_t length;
} HTTPResponse;
```

- Using **send()** send the content of **vptrResponse** back
- Send **length** bytes
- Don’t use strlen(), a buffer might not be NULL terminated

Task 5 – Answer the Request - Testing

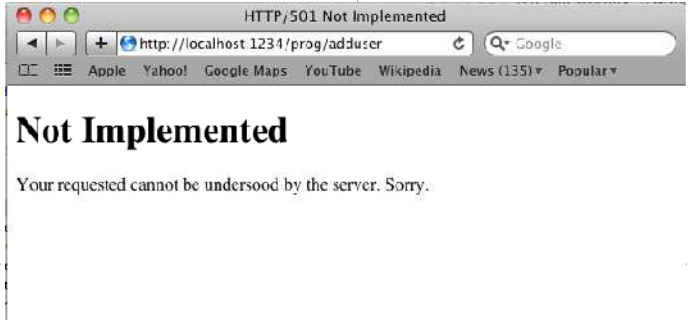
Listen
Threads
Request
Prepare the response
Send the response

- Using telnet:
  - Try to send the requests (in green) in the previous examples and verify if the HTTP response headers match with those presented there
- If it works, then start using a browser and enjoy!
  - Visit **`http://<hostname>:<port>/index.html`** and follow the links on that page


HTTP/404 File Not Found
http://localhost:1234/nosuchfile
Google
Apple Yahoo! Google Maps YouTube Wikipedia News (135) Popular
http://maps.google.com/

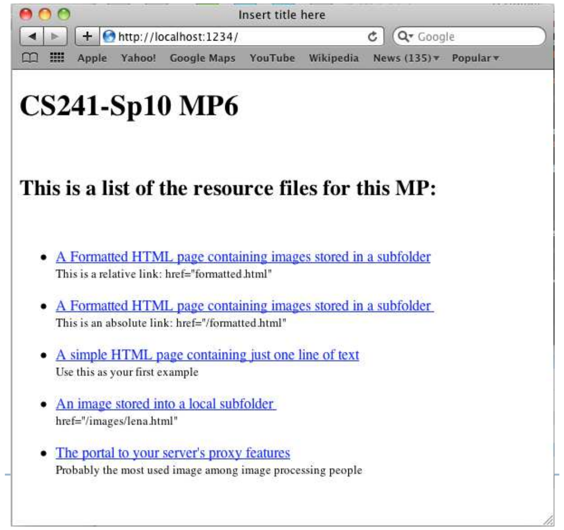
Not Found

Your requested file was not found on the server.


HTTP/501 Not Implemented
http://localhost:1234/prog/adduser
Google
Apple Yahoo! Google Maps YouTube Wikipedia News (135) Popular

Not Implemented

Your requested cannot be undersood by the server. Sorry.


Insert title here
http://localhost:1234/
Google
Apple Yahoo! Google Maps YouTube Wikipedia News (135) Popular

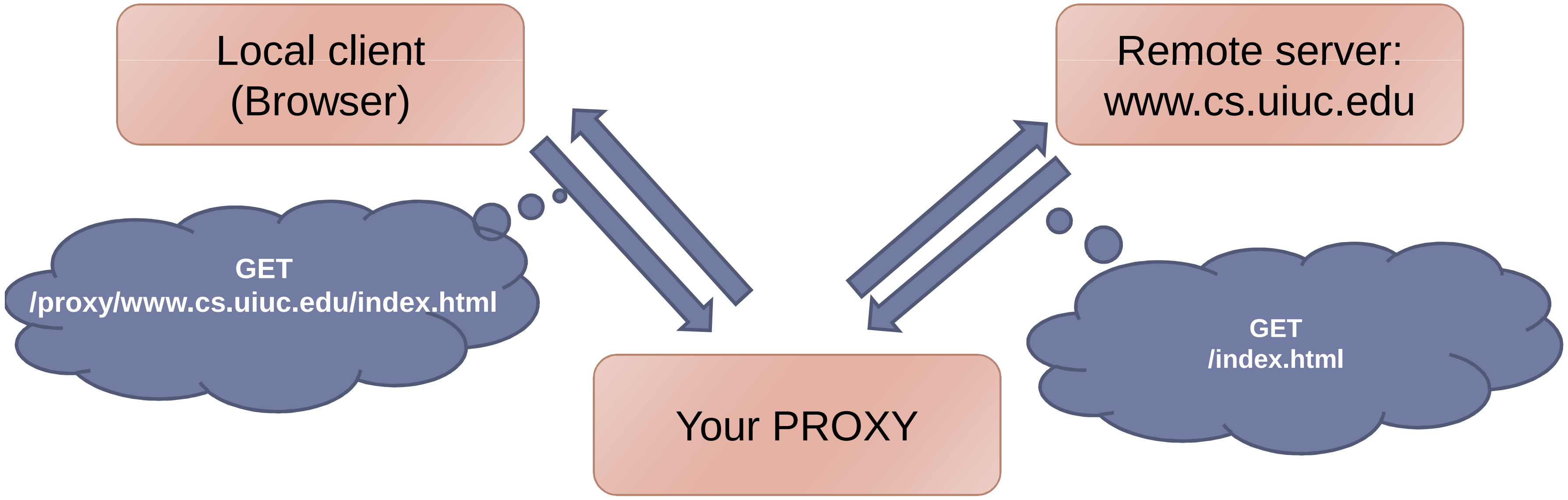
CS241-Sp10 MP6

This is a list of the resource files for this MP:

- A Formatted HTML page containing images stored in a subfolder
  This is a relative link: href="formatted.html"
- A Formatted HTML page containing images stored in a subfolder
  This is an absolute link: href="/formatted.html"
- A simple HTML page containing just one line of text
  Use this as your first example
- An image stored into a local subfolder
  href="/images/lena.html"
- The portal to your server's proxy features
  Probably the most used image among image processing people

Task 6 – Proxy!

- A proxy is a man in the middle that forwards the requests to a remote server and the responses to the local client:


Local client (Browser)
Remote server: www.cs.uiuc.edu
GET /proxy/www.cs.uiuc.edu/index.html
GET /index.html
Your PROXY

Task 6 – Proxy!

Listen
Threads
Request
Prepare the response
Send the response
Proxy requests

- A GET request for the proxy looks like:

```
GET  /proxy/www.cs.uiuc.edu/class/sp10/cs241/index.html HTTP/1.1
```

- Split into:
  - `/proxy` → marks the proxy request
  - `www.cs.uiuc.edu` → the remote server we must connect to
  - `/class/sp10/cs241/index.html` → the remote file that must be retrieved

- Make an HTTP request to the remote server and forward the response to the client

Task 6 – Proxy!

- A GET request for the proxy looks like:

```
GET  /proxy/www.cs.uiuc.edu/class/sp10/cs241/index.html HTTP/1.1
```

What if the remote server does not exist?

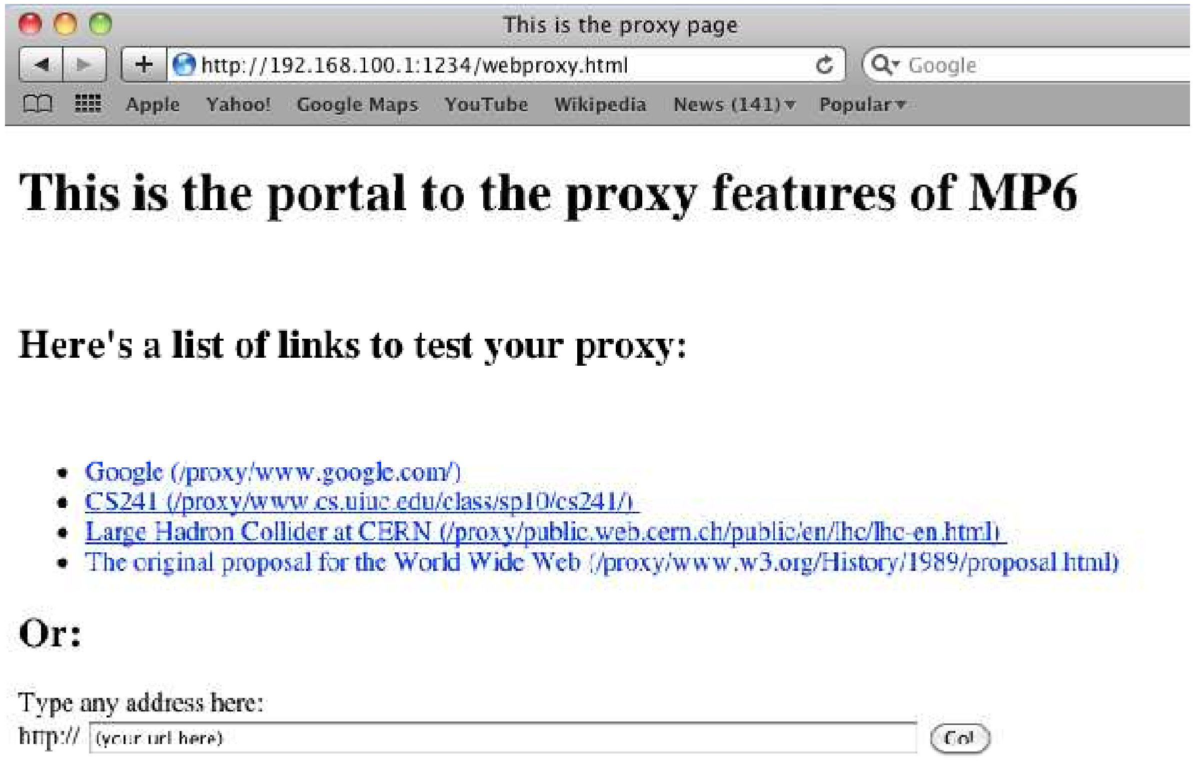
Just make sure that your proxy does not crash (sending a 404 error message is not required but it would be nice)

- Make an HTTP request to the remote server and forward the response to the client

Task 6 – Proxy! - Testing

Listen
Threads
Request
Prepare the response
Send the response
Proxy requests

- We provide some html page in your mp6 folder. Check `webproxy.html`



This is the portal to the proxy features of MP6

Here's a list of links to test your proxy:

- Google (/proxy/www.google.com/)
- CS241 (/proxy/www.cs.uiuc.edu/class/sp10/cs241/)
- Large Hadron Collider at CERN (/proxy/public.web.cern.ch/public/en/lhc/lhc-en.html)
- The original proposal for the World Wide Web (/proxy/www.w3.org/History/1989/proposal.html)

Or:

Type any address here:
http:// (your url here) Go!

A Break: HTML tags

- An HTML page is a plain text file:
  - Browsers print part of the text in their windows
  - Part of it are interpreted as commands (tags)

- Some comong tags:
  - **`<b> TEXT </b>`** → the text is printed in bold font
  - **`<i> TEXT </i>`** → the text is printed in italic
  - **`<h1> TEXT </h1>`** → use the header1 style

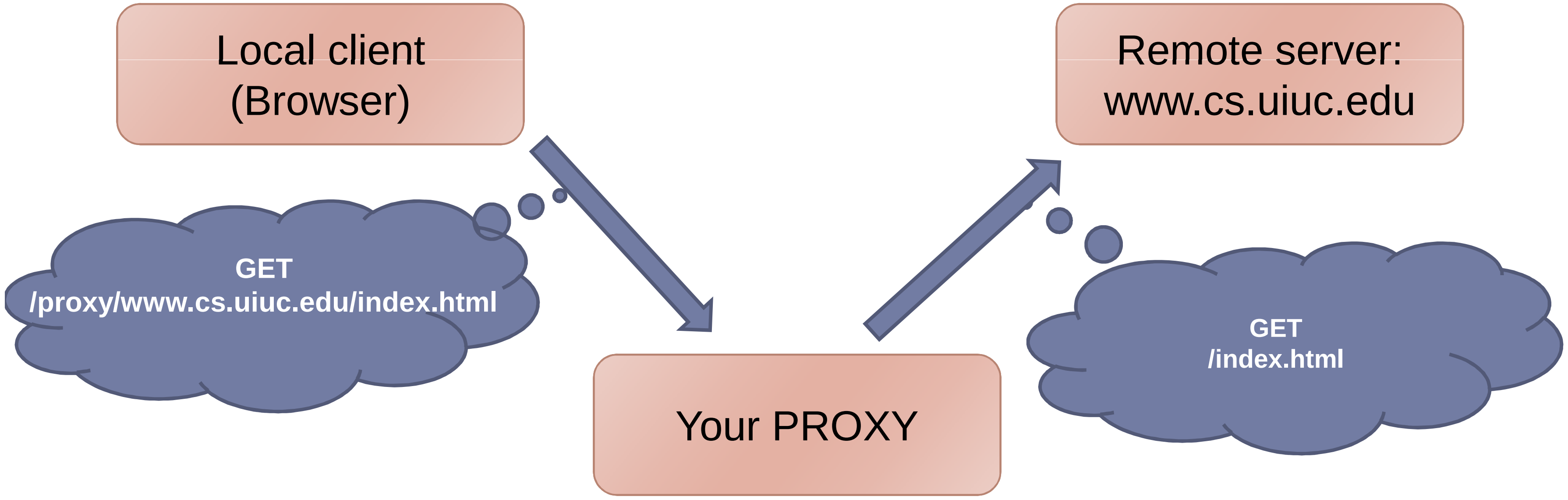
A Break: HTML tags

- Some tags have pointers to other resources:
  - **`<a href=“someurl“> TEXT </b> :`**
    - TEXT is a link
    - If you click on TEXT the browser will fetch the url defined in the href= property
    - The <a> tags have many properties, href is one of them. There can be a variable number of properties in any order
  - **`<img src=“someurl”>`**:
    - The browser fetches the image pointed by someurl and displays it
    - Here too, src might be anywhere between **`<img`** and **`>`**

Task 7 – Rewriting the Links

Listen
Threads
Request
Prepare the response
Send the response
Proxy requests
Rewrite links

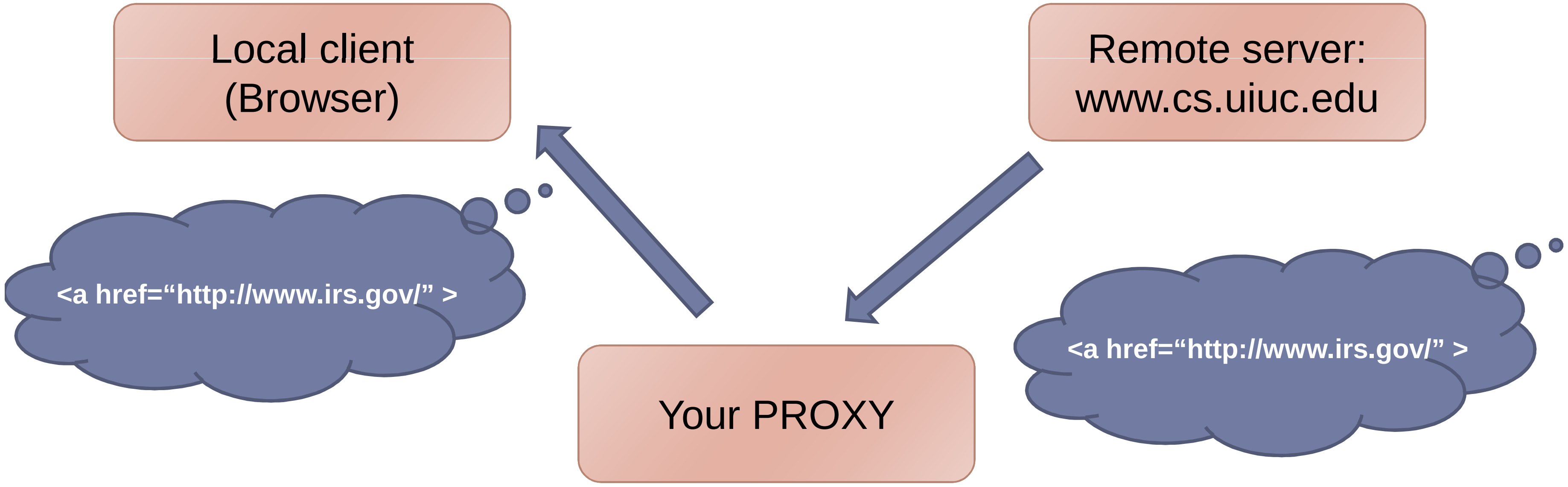
- Why do we need to rewrite links?
- When we make a request for a remote server we go through the proxy


Local client (Browser)
GET /proxy/www.cs.uiuc.edu/index.html
Your PROXY
Remote server: www.cs.uiuc.edu
GET /index.html

Task 7 – Rewriting the Links

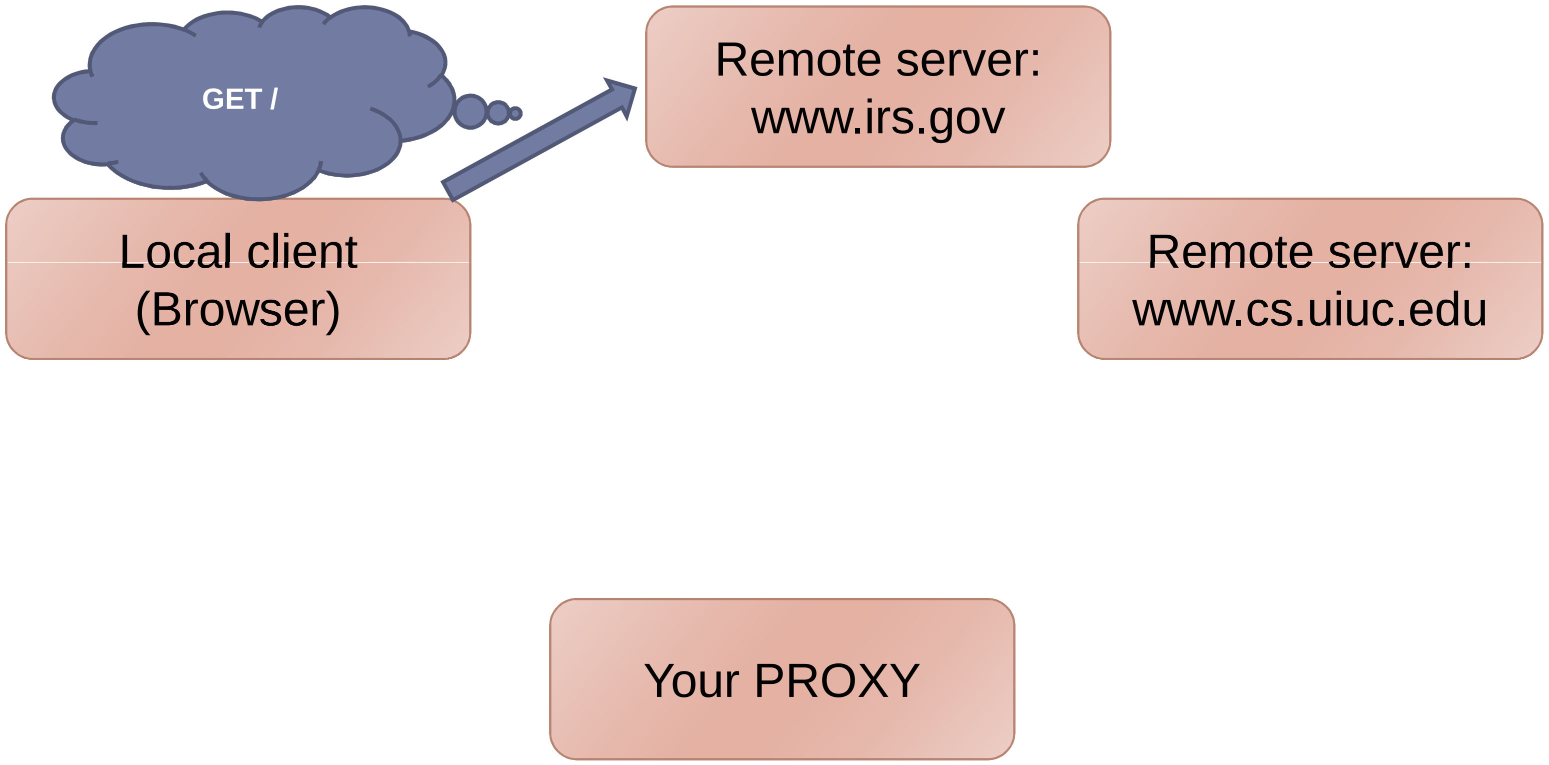
Listen
Threads
Request
Prepare the response
Send the response
Proxy requests
Rewrite links

- Why do we need to rewrite links?
- What if the html file we retrieve has some link to other pages?


Local client (Browser)
Remote server: www.cs.uiuc.edu
<a href="http://www.irs.gov/" >
<a href="http://www.irs.gov/" >
Your PROXY

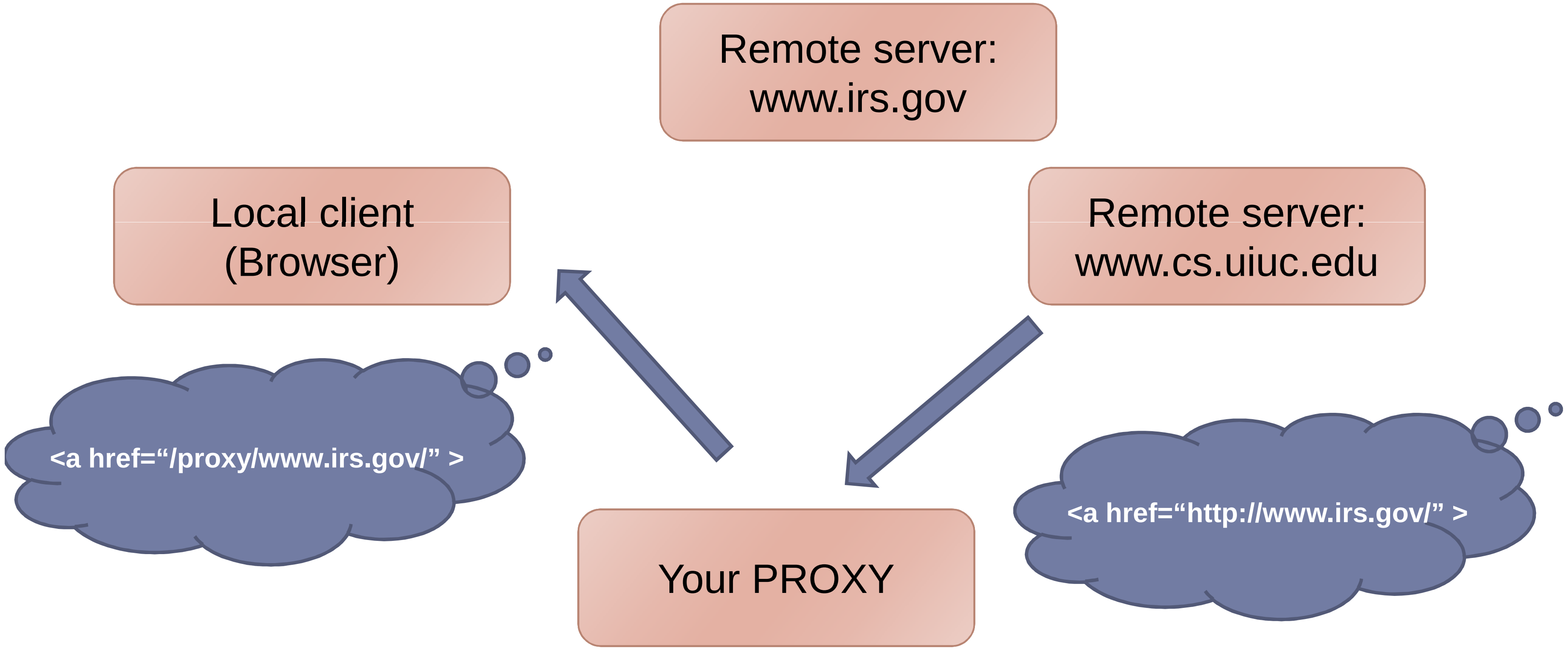
Task 7 – Rewriting the Links

- Clicking on the link we would skip the proxy!!!



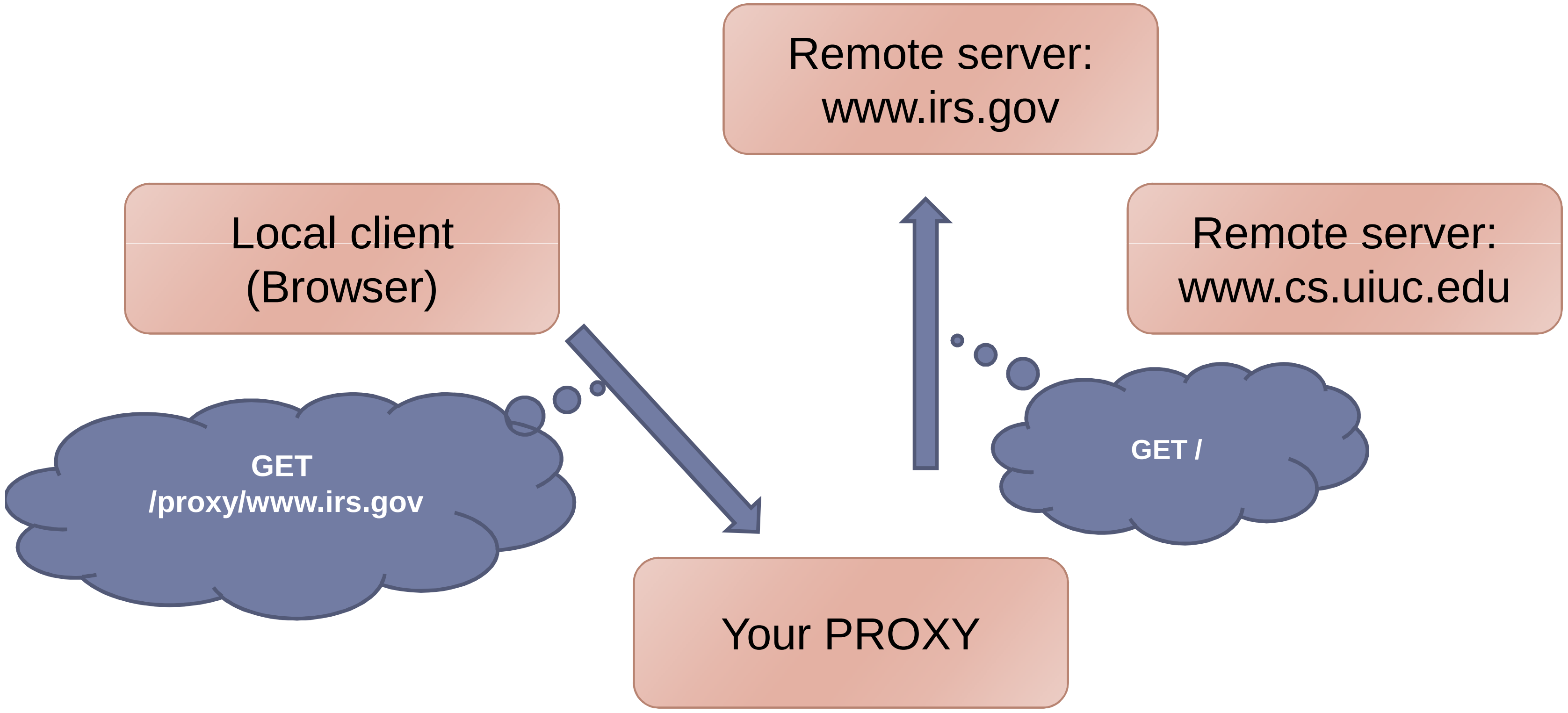
Task 7 – Rewriting the Links

- If instead we rewrite the link:

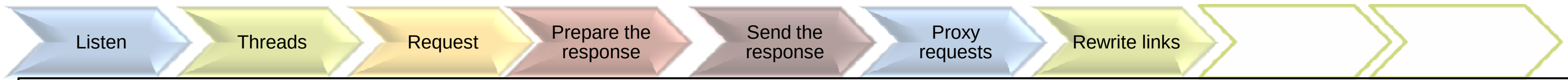

Remote server: www.irs.gov
Local client (Browser)
Remote server: www.cs.uiuc.edu
<a href="/proxy/www.irs.gov/" >
<a href="http://www.irs.gov/" >
Your PROXY

Task 7 – Rewriting the Links

- The second request goes to the proxy as well!



Task 7 – Rewriting the Links



- Rewrite:
  - <a […] href="URL"> → hyperlinks to other pages
  - <img […] src="URL"> → images to be displayed on the page

- When answering to a request for **`/proxy/host1/file.html:`**
  - Rewrite only Absolute links:
    - http://newhost/path → /proxy/newhost/path
    - /absolute path → /proxy/host1/absolutepath
  - Do not rewrite relative links:
    - Path without leading "/"

Task 7 – Rewriting the Links - Testing

Listen
Threads
Request
Prepare the response
Send the response
Proxy requests
Rewrite links

- Check the source code of the pages that you visit from `webproxy.html`, they should have been rewritten
- In most browsers, if you move your mouse over a link you can see in the status bar what that link points to

Task 8 – Keepalive Connections

Listen
Threads
Request
Prepare the response
Send the response
Proxy requests
Rewrite links
KeepAlive connections

- If the browser request had the line:
  - `Connection: close`
    - After your thread has server the request, close the socket and return
  - `Connection: keep-alive`
    - After your thread has served the request, call recv() again and wait for a new request
    - The browser could also close the file descriptor later, in this case recv() returns 0

Task 9 – SIGINT

Listen
Threads
Request
Prepare the response
Send the response
Proxy requests
Rewrite links
KeepAlive connections
SIGINT

- Write a **`SIGINT`** handler
  - To send it to your proxy hit CTRL-C in the console where proxy is running.
  - The handler should initiate a process that frees all the memory and stop waiting for incoming connections
  - Once all threads currently serving a request are done, your program should exit

You are Done!

Listen
Threads
Request
Prepare the response
Send the response
Proxy requests
Rewrite links
KeepAlive connections
SIGINT

- That’s all, your proxy should be up and running
- Use the local pages that we provide in your MP folder to test your proxy
- Those pages contain a lot of different links that should test most (but not all!) possible cases. Make your own!