HTTP

Hypertext Transfer Protocol (HTTP) is a communications protocol for the transfer of information on internets and the World Wide Web. Its original purpose was to provide a way to publish and retrieve hypertext pages over the Internet. HTTP can be implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport.
HTTP is a request/response standard between a client and a server. A client is the end-user, the server is the web site. The client making an HTTP request - using a web browser, spider, or other end-user tool - is referred to as the user agent. The responding server - which stores or creates resources such as HTML files and images - is called the origin server.

To know more about HTTP protocol you can read up the following links:

http://www.w3.org/Protocols/rfc2616/rfc2616.html
http://www.lincoln.edu/math/rmyrick/ComputerNetworks/InetReference/102.htm

In this assignment, you first need to implement a simple HTTP protocol by developing a Client and a Server which communicates through HTTP-type messages over a TCP connection. The server is a TCP concurrent server. The client will send HTTP requests to the server. The server will parse the messages and respond accordingly.

An HTTP request consists of a method name describing the operation, a resource name (an URI) describing the object in which the method is to be applied, and optional data. For ease of completion in this assignment you must implement only the following HTTP method:

GET: Retrieves the specified resource from the server. (ex. the contents of a file).

You can read more about HTTP request messages here:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5

In the next part, you need to implement a HTTP Proxy that can act as a proxy server between the client and the server. Further along the line, you will implement a very simple implementation of server load balancing called Round - Robin DNS.

The details of the you have to implement are as follows. The format of HTTP requests and responses are also given in these details. Please note that although we are trying to implement a subset of the HTTP protocol set, the protocol must be strictly adhered to. We will be testing your client against a standard HTTP proxy server, and then test the server using a standard browser. Hence, again, please make sure these programs follow the protocol strictly.

While implementing the DNS Load balancing, you will need to setup a domain name for the server. Since the dns requests will usually go to our server, we can come up with any name, as long as the dns server returns an IP for that domain.

HTTP Server:

The web server you will implement is a simple file server.
A sample URL like http://www.customwebsite.org:53007/index.html has 4 parts,

- the protocol specifier (http://),
- the server domain name (www.customwebsite.org),
- the port of the web server (53007),
- and the file to be retrieved (index.html).

When a server starts running, the current directory is treated as the document root, so assume that if the server starts on a folder, all documents are relative to it. On receiving a HTTP request from the client the server parses through the request, prepares the reply message (it can also make an error message as reply) and sends it to the client who requested for it. The format of the reply message is described in the Client section. The Server must print on the command window every request message it receives (in exact format received).

HTTP Client:

The client is a command line based web browser. The format of an actual http request message is given here:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.

However, for simplicty, we will use a simpler format for the request message. It is similar to an HTTP request, but does not have many of the fields that are not required for us.The format is as follows:

```
Request = Request-Line CRLF
          (Request-Header CRLF)*
          CRLF
          Message-Body
```

```
Request-Line = Method-Name <space> Request-URI <space> HTTP/1.0 CRLF
Request-Header = Header-Name ":" <space> Header-Content CRLF
```

Method-Name can be GET, HEAD, POST, PUT, DELETE, etc. In this assignment we will be only considering the GET method. Every HTTP request/response line must be terminated using a CRLF (carriage return, line feed) which can be represented in C/C++ as the 2 characters "\r\n"

The method to be implemented here is GET. <space> stands for a single space character. Request-URI is the URI of the resource on which the method is to be applied. We send HTTP/1.0 as we try to implement the HTTP/1.0 specifications.

The Request-Line is followed by 0 or more HTTP Headers. These Request-Headers are of the format <name>: <content> CRLF
For a simple GET request, assume that the client will further send the following headers:
Host: <domain-name-of-server>
User-Agent: <a-string-that denotes your user-agent, can be anything>
Connection: close

For POST, PUT and some other methods, an additional Content-Length: <length-in-bytes> header is also added. In that case the HTTP request will contain a message body, whose size is specified by the Content-Length header. The message body contains the actual optional data.
Hence, HTTP headers will always end with a double sequence of CRLF, i.e. "\r\n\r\n".

A server response message will have the following format:

```
Response = Status-Line CRLF
           (Response-Header CRLF)*
           CRLF
           [message-body]
```

```
Status-Line = HTTP/1.0 <space> Status-Code <space> Reason-Phrase CRLF
```

The mandatory HTTP Headers in this case are :-
Accept-Ranges: bytes
Content-Length: <message-body-length>
Content-Type: text/html; charset=UTF-8
Connection: close

Status-Code is always an integer. In case the document is found, it is returned with Status-Code = 200, Reason-Phrase = "OK"
If the document is missing, we issue a 404 error, namely Status-Code = 404, Reson-Phrase = "Not Found". In case the document is not found, the server must generate a message body which can be something similar to :

<html><body>The document could not be found.</body></html>

To read about HTTP status codes, follow this link : http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

"Connection: close" is a header that we will always send along with the standard HTTP headers, this header specifies that the client does not wish to re-use the connection. HTTP/1.1 specification has a pipelining mode where a single HTTP client can request multiple documents (http://en.wikipedia.org/wiki/HTTP_persistent_connections). To disable pipelining and yet confirm to the specs, we add this additional header to every request and to every response.

HTTP Proxy:

The HTTP Proxy server is a HTTP client and server both in one. It acts as a server to the Client, but as a client to the Server. When the client requests a document from the proxy server, the proxy server must make a connection to the Server, retrieve the document and send it back to the Client.

A Proxy Server usually runs on port 8080. In this case we will use the server port as (5000 + roll. no. of 2nd team member)

A Proxy Server must confirm to the following protocol, which is a reduced version of :-

- Client(C) wants to retrieve http://www.customwebsite.org:53007/image.jpg via the Proxy Server(P).
- C does not DNS resolve **www.customwebsite.org** to an IP address.
- C connects to P. Let this socket connection be called cli_sockfd on the server side.
- C sends the exact HTTP Headers as if connecting directly to the Server(S), except for a slight modification:
  Since C needs to tell P that the server(S) is example.org, instead of
  GET /image.jpg HTTP/1.0
  it sends
  GET http://www.customwebsite.org:53007/image.jpg HTTP/1.0
  i.e. the full path of the link. Note that the domain name is sent instead of the IP address.

- P waits till the client has sent all the data.
- On completion, P first parses the headers, and then finds out the server link. It then parses the domain www.example.org and resolves it to the IP address of the Server. Note hence that domain resolution happens on the proxy-server side, unlike that in the case of a direct connection. If domain resolution fails, show a 404 error page like that of the HTTP server and close the connection cli_sockfd.
- P now acts as the client. It connects to the Server S, via a socket connection serv_sockfd. This time it sends the same headers as the client C did, but emulating that of a client. (Hence, the Server never knows whether it was a direct connection or via a Proxy).
- P sends the headers, but modifying the GET request to change it to **GET /image.jpg HTTP/1.0**.
- P receives the reply back, which is HTTP headers and the document body.P closes serv_sockfd.
- P again toys with the HTTP headers, adding another additional header, named Via. This denotes that the request came from the proxy server. The Via header is as follows:

- **Via: <Proxy-Server-Name>**
- The HTTP headers and the body are sent in the same format as that in a HTTP Server. After that, P closes cli_sockfd.

DNS Load Balancing:-

DNS Load Balancing is a popular feature to reduce server load. Wikipedia link: http://en.wikipedia.org/wiki/Round_robin_DNS
We will implement DNS load balancing using our custom dns client server. You need to :-

1. Implement the HTTP client and proxy-server to perform all DNS lookups using the DNS client developed in Assignment 2. Assume only a UDP DNS Server is present.
2. Modify the UDP DNS Server. Make sure that a request to resolve the domain for the server resolves to more than 1 IP. For example, if the domain name for this assignment is www.customwebsite.org, whenever the DNS Server requests a domain resolution for www.customwebsite.org, it should return 3 IP addresses which have been hardcoded in the IP server, without performing the gethostbyname().
3. These IP addresses must be returned to the client in a round-robin fashion.

An example is shown below:

Custom Domain: www.customwebsite.org
IP address configured in server: 10.341.2.5, 10.341.2.6, 10.341.10.15
Client 1 makes a request. IP addresses returned (note order) -
10.341.2.5, 10.341.2.6, 10.341.10.15
Client 2 makes a request. IP addresses returned (note order) -
10.341.2.6, 10.341.10.15, 10.341.2.5
Client 3 makes a request. IP addresses returned (note order) -
10.341.10.15, 10.341.2.5, 10.341.2.6
Client 4 makes a request. IP addresses returned (note order) -
10.341.2.5, 10.341.2.6, 10.341.10.15
and so on ...

Query based Load Balancing:

to be added -Arindam 19/08/2009 06:31

Linix netcat command, and the HTTP transaction example

Copied from the linux manpages:

> **netcat** is a simple unix utility which reads and writes data across network connections, using TCP or UDP protocol. It is designed to be a reliable "back-end" tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities. Netcat, or "nc" as the actual program is named, should have been supplied long ago as another one of those cryptic but standard Unix tools.
>
> In the simplest usage, "netcat host port" creates a TCP connection to the given port on the given target host. Your standard input is then sent to the host, and anything that comes back across the connection is sent to your standard output. This continues indefinitely, until the

network side of the connection shuts down. Note that this behavior is different from most other applications which shut everything down and exit after an end-of-file on the standard input.

Netcat can also function as a server, by listening for inbound connec-tions on arbitrary ports and then doing the same reading and writing. With minor limitations, netcat doesn't really care if it runs in "client" or "server" mode -- it still shovels data back and forth until there isn't any more left. In either mode, shutdown can be forced after a configurable time of inactivity on the network side.

The following is a simple HTTP transaction I have performed via netcat. The Data sent from the client is written in black, while the data returned by the server is in blue.

Direct Connection Transaction (You might want to try changing it to iitkgp.ac.in to see how it works via a direct connection, I did not add it since the document body is too long) :-

```
arindam@Numenor:~$ nc www.example.org 80
GET /index.html HTTP/1.0
Host: www.example.org
User-Agent: My Awesome HTTP Client
Connection: close

HTTP/1.1 200 OK
Date: Wed, 19 Aug 2009 00:36:01 GMT
Server: Apache/2.2.3 (Red Hat)
Last-Modified: Tue, 15 Nov 2005 13:24:10 GMT
ETag: "b80f4-1b6-80bfd280"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8

<HTML>
<HEAD>
  <TITLE>Example Web Page</TITLE>
</HEAD>
<body>
<p>You have reached this web page by typing &quot;example.com&quot;,
&quot;example.net&quot;,
  or &quot;example.org&quot; into your web browser.</p>
<p>These domain names are reserved for use in documentation and are not
available
  for registration. See <a href="http://www.rfc-editor.org/rfc/
rfc2606.txt">RFC
  2606</a>, Section 3.</p>
</BODY>
</HTML>
```

Proxy Connection Transaction (This actually works from KGP LAN :) -

```
arindam@Numenor:~$ nc 144.16.192.247 8080
GET http://www.example.org/index.html HTTP/1.0
Host: www.example.org
User-Agent: My Awesome HTTP Client
Connection: close
```

```
HTTP/1.0 200 OK
Date: Tue, 18 Aug 2009 23:43:11 GMT
Server: Apache/2.2.3 (Red Hat)
Last-Modified: Tue, 15 Nov 2005 13:24:10 GMT
ETag: "b80f4-1b6-80bfd280"
Accept-Ranges: bytes
Content-Length: 438
Content-Type: text/html; charset=UTF-8
Age: 3527
X-Cache: HIT from proxy245.iitkgp.ernet.in
X-Cache-Lookup: HIT from proxy245.iitkgp.ernet.in:8080
X-Cache: MISS from proxy247.iitkgp.ernet.in
X-Cache-Lookup: MISS from proxy247.iitkgp.ernet.in:8080
Via: 1.1 proxy245.iitkgp.ernet.in:8080 (squid/2.7.STABLE3), 1.0
proxy247.iitkgp.ernet.in:8080 (squid/2.7.STABLE3)
Connection: close

<HTML>
<HEAD>
  <TITLE>Example Web Page</TITLE>
</HEAD>
<body>
<p>You have reached this web page by typing &quot;example.com&quot;,
&quot;example.net&quot;,
  or &quot;example.org&quot; into your web browser.</p>
<p>These domain names are reserved for use in documentation and are not
available
  for registration. See <a href="http://www.rfc-editor.org/rfc/
rfc2606.txt">RFC
  2606</a>, Section 3.</p>
</BODY>
</HTML>
```

Hints, Checklists and Assumptions:

- Assume that the body length never exceeds 5kB (i.e. 5 * 1024 bytes).
- Assume that the headers are not over 10 in number, and each will header size is less than 1024 characters.
- The Http Server and the proxy server must be concurrent servers.
- The client may request binary data from the server. Make sure that binary data, even something containing NULL characters, can be transacted faithfully.
- A client like the browser may send a HTTP/1.1 request instead of HTTP/1.0 in any case, ignore that and reply back with a HTTP/1.0 anyway.
- An external client like a browser can send more headers than the ones mentioned in the assignment. Similarly, an external server sends additional headers. Ignore them, but make sure the server/client do not crash if they appear.

Submission:

Submit a zipped file (tar.gz or zip preferably) which contains the code and a Makefile.

Running make in that folder should create the following executables :-
http_server
dns_server
http_proxy
http_client

The executables must take the following command line parameters :-
1. dns_server:
./dns_server <custom-domain-name> <ip-address-1> <ip-address-2> ... <ip-address-n>
example:
./dns_server www.customwebsite.org 10.341.2.5, 10.341.2.6, 10.341.10.15

2. http_server:
./http_server <bind-port-number>
example:
./http_server 53007

3. http_proxy:
./http_proxy <bind-port-number>
example:
./http_proxy 53011

4. http_client
It takes the following parameters:
URL to receive
filename to save the file as
proxy or direct connection to be used
./http_client <URL> <filename> <proxy>
URL is of the form : http://www.customwebsite.org:53007/index.html
filename is, say: save-file.txt
proxy can be either direct (using a direct connection)
or a proxy server, in the format : 144.16.192.247:8080

Example (direct)-
./http_client http://www.customwebsite.org:53007/image.jpg picture.jpg direct
Example (proxy)-
./http_client http://www.customwebsite.org:53007/random.pdf 404-page.htm 10.33.14.20:53011

You may use C or C++ for coding the assignment.