Computer Networks (CS-307)

Fall 2014

Assignment#2

Total Marks 75 = 50 +25

Deadline: 2nd October, 2014

Dear students this assignment has two parts:

- In Part - 1 you are required to create HTTP Proxy Server for NU Network.
- Part – 2 Problems from Exercise 2 of 6th Edition.

SUBMISSION INSTRUCTIONS FOR PART - 1

- Submission deadline for Part-1 is **Thursday, 2nd October, 2014 until 6:00 pm on slate. Y**ou should paste all of your **code must be in a single .txt file** and submit it using SLATE in a timely manner.
- Submission in any other format (.cpp, rar, zip) will **NOT** be accepted
- If you didn't followed the submission instructions you will be heavily penalized.
- Code File name should be Roll No_Sec e.g. 12i-1111_A
- Late submissions are **NOT** accepted.
- Make sure your programs do not crash when given bad input, but instead provide warning messages
- **Copy** cases will get a **'Zero'** in all Assignments.
- Marks will be allocated for code style. This includes appropriate use of comments and indentation for readability, plus good naming conventions for variables, constants, and functions. Your code should also be well structured (i.e. not all in the main function).

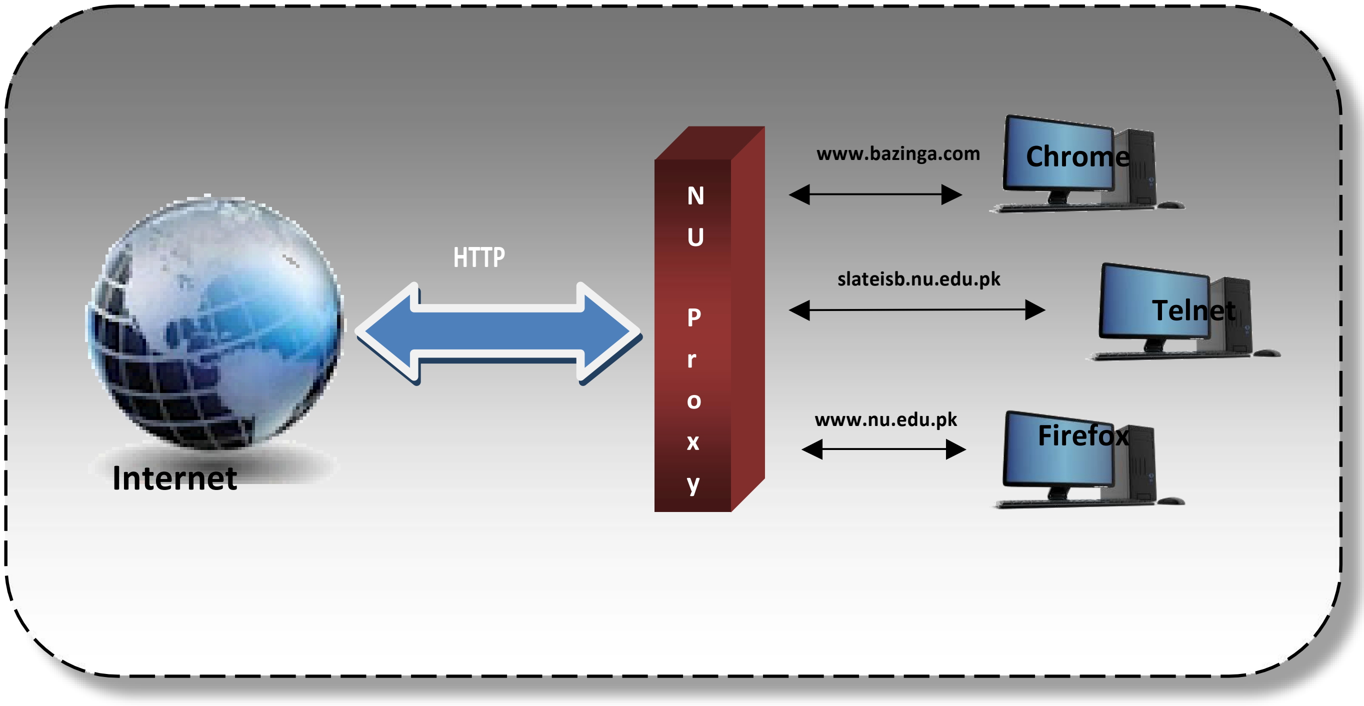
SUBMISSION INSTRUCTIONS FOR PART - 2

- Submission deadline for Part - 2 is **Thursday, 2nd October, 2014 until 2:00 pm in instructor's offices.**Problems of Part – 2 **should be** hand written on A4 size papers.
- **Copy** cases will get a **'Zero'** in all Assignments.

PART – 1

Building an HTTP Proxy for NU Network

Individual Assignment



Problem Statement

In this assignment, you will implement a simple web proxy for NU LAN that passes requests and data between a web client and a web server. This will give you a chance to get to know one of the most popular application protocols on the Internet- the Hypertext Transfer Protocol (HTTP) v. 1.0 - and give you an introduction to the Berkeley sockets API. When you're done with the assignment, you should be able to configure your web browser to use your personal proxy server as a web proxy.

Introduction: The Hypertext Transfer Protocol

The Hypertext Transfer Protocol or (HTTP) is the protocol used for communication on this web. That is, it is the protocol, which defines how your web browser requests resources from a web server and how the server responds. For simplicity, in this assignment we will be dealing only with version 1.0 of the HTTP protocol, defined in detail in RFC 1945. You should read through this RFC and refer back to it when deciding on the behavior of your proxy.

HTTP communications happen in the form of transactions, a transaction consists of a client sending a request to a server and then receiving the response. Request and response messages share a common basic format:

- An initial line (a request or response line, as defined below)
- Zero or more header lines
- A blank line (CRLF)
- An optional message body.

For most common HTTP transactions, the protocol boils down to a relatively simple series of steps (important sections of RFC 1945 are in parenthesis):

1. A client creates a connection to the server.
2. The client issues a request by sending a line of text to the server. This request line consists of a HTTP method (most often GET, but others are defined in the RFCs), a request URI (like a URL), and the protocol version that the client wants to use (HTTP/1.0). The message body of the initial request is typically empty. (5.1-5.2, 8.1-8.3, 10, D.1)
3. The server sends a response message, with its initial line consisting of a status line, indicating if the request was successful. The status line consists of the HTTP version (HTTP/1.0), a response status code (a numerical value that indicates whether or not the request was completed successfully), and a reason phrase, an English-language message providing description of the status code. Just as with the the request message, there can be as many or as few header fields in the response as the server wants to return. Following the CRLF field separator, the message body contains the data requested by the client in the event of a successful request. (6.1-6.2, 9.1-9.5, 10)
4. Once the server has returned the response to the client, it closes the connection.

It's fairly easy to see this process in action without using a web browser. From a Unix prompt, type:

```
telnet www.yahoo.com 80
```

This opens a TCP connection to the server at www.yahoo.com listening on port 80- the default HTTP port. You should see something like this:

```
Trying 209.131.36.158...
Connected to www.yahoo.com (209.131.36.158).
Escape character is '^]'.
```

type the following:

```
GET / HTTP/1.0
```

and hit enter twice. You should see something like the following:

```
HTTP/1.1 200 OK
Date: Fri, 10 Nov 2006 20:31:19 GMT
Connection: close
Content-Type: text/html; charset=utf-8

<html><head>
<title>Yahoo!</title>
(More HTML follows)
```

There may be some additional pieces of header information as well- setting cookies, instructions to the browser or proxy on caching behavior, etc. What you are seeing is exactly what your web browser sees when it goes to the Yahoo home page: the HTTP status line, the header fields, and finally the HTTP message body- consisting of the HTML that your browser interprets to create a web page.

HTTP Proxies

Ordinarily, HTTP is a client-server protocol. The client (usually your web browser) communicates directly with the server (the web server software). However, in some circumstances it may be useful to introduce an intermediate entity called a proxy. Conceptually, the proxy sits between the client and the server. In the simplest case, instead of sending requests directly to the server the client sends all its requests to the proxy. The proxy then opens a connection to the server, and passes on the client's request. The proxy receives the reply from the server, and then sends that reply back to the client. Notice that the proxy is essentially acting like both a HTTP client (to the remote server) and a HTTP server (to the initial client).

Why use a proxy?

There are a few possible reasons:

- **Performance:** By saving a copy of the pages that it fetches, a proxy can reduce the need to create connections to remote servers. This can reduce the overall delay involved in retrieving a page, particularly if a server is remote or under heavy load.
- **Content Filtering and Transformation:** While in the simplest case the proxy merely fetches a resource without inspecting it, there is nothing that says that a proxy is limited to blindly fetching and serving files. The proxy can inspect the requested URL and selectively block access to certain domains, reformat web pages (for instances, by stripping out images to make a page easier to display on a handheld or other limited-resource client), or perform other transformations and filtering.
- **Privacy:** Normally, web servers log all incoming requests for resources. This information typically includes at least the IP address of the client, the browser or other client program that they are using (called the User-Agent), the date and time, and the requested file. If a client does not wish to have this personally identifiable information recorded, routing HTTP requests through a proxy is one solution. All requests coming from clients using the same proxy appear to come from the IP address and User-Agent of the proxy itself, rather than the individual clients. If a number of clients use the same proxy (say, an entire business or university), it becomes much harder to link a particular HTTP transaction to a single computer or individual.

Links

- RFC 1945 The Hypertext Transfer Protocol, version 1.0

Assignment Details

The Basics

Your first task is to build a basic web proxy capable of accepting HTTP requests, making requests from remote servers, and returning data to a client.

It should compile and run without errors producing a binary called `proxy` that takes as its first argument a port to listen from. Don't use a hard-coded port number.

You shouldn't assume that your server will be running on a particular IP address, or that clients will be coming from a pre-determined IP.

Listening

When your proxy starts, the first thing that it will need to do is establish a socket connection that it can use to listen for incoming connections. Your proxy should listen on the port specified from the command line, and wait for incoming client connections.

Once a client has connected, the proxy should read data from the client and then check for a properly-formatted HTTP request. An invalid request from the client should be answered with an appropriate error code.

Parsing the URL

Once the proxy sees a valid HTTP request, it will need to parse the requested URL. The proxy needs at most three pieces of information: the requested host and port, and the requested path.

Getting Data from the Remote Server

Once the proxy has parsed the URL, it can make a connection to the requested host (using the appropriate remote port, or the default of 80 if none is specified) and send a HTTP request for the appropriate file. The proxy then sends the HTTP request that it received from the client to the remote server.

Returning Data to the Client

After the response from the remote server is received, the proxy should send the response message to the client via the appropriate socket. Once the transaction is complete, the proxy should close the connection.

Performance and Design Considerations

- You must not use a hard-coded port number.
- You must be able to handle multiple simultaneous connections. You can implement this feature however you like- forking, threading, etc.
- You are only required to implement HTTP 1.0 methods. Additional methods specified in the HTTP 1.1 RFC should be responded to with a 501 error.
- You must support the basic HTTP 1.0 methods- GET, HEAD, and POST. Additional methods specified in the RFC can be implemented as an additional feature.
- Configuring a Web Browser to use YOUR proxy to be fetch a webpage.
- You do not need to implement checking of header fields. As long as they follow the basic format specified in the RFC (section 4.2), they should be passed through unmodified.

Testing Your Proxy

Run your client with the following command:

`./proxy <port>`, where `port` is the port number that the proxy should listen on. As a basic test of functionality, try requesting a page using telnet:

```
telnet localhost <port>
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
GET http://www.google.com HTTP/1.0
```

If your proxy is working correctly, the headers and HTML of the Google homepage should be displayed on your terminal screen.

For a slightly more complex test, you can configure your web browser to use your proxy server as its web proxy. See the section below for details.

Configuring a Web Browser to Use a Proxy

Using your proxy with a web browser will not work 100% correctly until you have multi-threading/forking working. Because a web browser like Firefox or IE issues multiple HTTP requests for each URL you request (for instance, to download images and other embedded content), a single-threaded proxy will likely miss some requests, resulting in missing images or other minor errors. That’s OK for testing, but you must correctly handle multiple connections in your final submission.

In browser settings, change proxy settings to your server’s IP address & port number for browser to start acting as a client.

Extra Credit Options

- Adding the “Prefetch” option to your proxy.
- Content Filtering

(Note: Extra credit will not be graded unless basic requirements have been completed.)

Socket Programming

In order to build your proxy you will need to learn and become comfortable programming sockets. The Berkeley sockets library is the standard method of creating network systems on Unix. There are a number of functions that you will need to use for this assignment:

- Parsing addresses:
    - inet_addr

  Convert a dotted quad IP address (such as 36.56.0.150) into a 32-bit address.

  gethostbyname

Convert a hostname (such as nu.edu.pk) into a 32-bit address.
getservbyname
Find the port number associated with a particular service, such as FTP.

- Setting up a connection:
    - socket

  Get a descriptor to a socket of the given type
  connect
  Connect to a peer on a given socket
  getsockname
  Get the local address of a socket

- Creating a server socket:
    - bind

  Assign an address to a socket
  listen
  Tell a socket to listen for incoming connections
  accept
  Accept an incoming connection

- Communicating over the connection:
    - read/write

  Read and write data to a socket descriptor
  htons, htonl / ntohs , ntohl
  Convert between host and network byte orders (and vice versa) for 16 and 32-bit values

You can find the details of these functions in the Unix `man` pages and Beej's Guide.

Grading

You should submit your completed proxy by the date posted on SLATE using the assignment submission page there. You will need to submit a single .txt file including all of your code:

Your proxy will be graded out of 50 points, with the following criteria:

1. Your assignment should create a binary name `proxy` that will compile and run without error. The first command line argument should be the port that the proxy will listen from. (7)
2. Your proxy should run silently- any status messages or diagnostic output should be off by default. Your proxy should create an output log file recording all web page requests (only requests not the whole contents). (10)
3. Fetching actual HTML pages by proxy server (10)
4. Any browser can be configured to use your proxy server to fetch pages (8)

5. You must support multiple connections. How you do this (fork, etc.) is up to you. (10)
6. Well written (good abstraction, error checking, and readability) and well commented code gets additional points (5).
7. There will also be extra score for the best extension to the proxy. Take a look at "Extra Credit Options" for some hints about possible extensions that you can add to the proxy.

PART – 2

Individual Assignment

Exercise 2 Problems

Attempt following Problems from exercise 2. [Note: Use 6th Edition for these problems]

P1, P3, P4, P5, P7, P6, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20 and P21.