EX2 – HTTP client

Goals:

The purpose of this project is two-fold:
1) give students hand-on experience with socket programming, and
2) help students better understand application-level protocols by implementing a well-known protocol—HTTP. In this programming assignment you will write HTTP client. Students are not required to implement the full HTTP specification, but only a very limited subset of it.

You will implement the following:

A HTTP client that constructs an HTTP request based on user's command line input, sends the request to a Web server, receives the reply from the server, and displays the reply message on screen. You should support only IPv4 connections.

Background:

What is HTTP?
HTTP stands for Hyper Text Transfer Protocol and is used for communication among web clients and servers. HTTP has a simple stateless client/server paradigm. The web client initiates a conversation by opening a connection to the server. Once a connection is set up, the client sends an HTTP request to server. Upon receiving the HTTP request from the client, the server sends an HTTP response back to the client. An HTTP request consists of two parts: a header and a body. In this project, HTTP request from a client doesn't contain a body. The first line of any request header should be:

Method Request-URI Version. An example HTTP1.1 request is:

GET /index.html HTTP/1.1
Host: www.jce.ac.il

The request header and body are separated by two sets of carriage return and line feed. Since we do not need the body, the end of a header marks the end of a request. Using a C char string, the example request above should be: "GET /index.html HTTP/1.1\r\nHost: www.jce.ac.il\r\n\r\n".

What is a URL?

Uniform Resource Locators (URLs) are formatted strings that identify resources in the web: documents, images, downloadable files, electronic mailboxes, etc. It generally has the format: Protocol://Host[:port]/Filepath. In this project, when a port is not specified, the default HTTP port number of 80 is used. For example, a file called "foo.html" on HTTP server "www.yoyo.com" in directory "/pub/files" corresponds to this URL: http://www.yoyo.com/pub/files/foo.html. The default HTTP network port is 80; if a HTTP server resides on a different network port (say, port 1234), then the URL becomes: http://www.yoyo.com:1234/pub/files/foo.html.

Program Description and What You Need to Do:

You will write the program client.c.

The Client

Command line usage: client [−h] [−d <time-interval>] <URL>. The flags and the url can come at any order, the only limitation is that the time interval should come right after the flag −d.

The client takes two options "-h" and "-d" and a required argument <URL>.

<URL> specifies the URL of the object that the client is requesting from server. The URL format is http://hostname[:port]/filepath.

Option "-h" specifies that the client only wants the response headers to be sent back from server. You should use HEAD method in your HTTP request when "-h" is specified in the command line.

Option "-d" along with its argument <time-interval> specify that the client wants the object only if it was modified in the past <time-interval>. <time-interval> is in the format "day:hour:minute" For example, if "-d 1:2:15" is included in the command line, then the client wants to get the object only if it was modified in the past 1 day, 2 hours and 15 minutes. When -d option is specified, the client should include an "If-Modified-Since:" header in the HTTP request and the value of this header is computed using the current time and the <time-interval> argument. In order to construct the time format for the ' If-Modified-Since ' header, you can use the following:

```
#define RFC1123FMT "%a, %d %b %Y %H:%M:%S GMT"
time_t now;
```

```
char timebuf[128];
now = time(NULL);
now=now-(day*24*3600+hour*3600+min*60); //where day, hour and min are the values
                                        //from the input
strftime(timebuf, sizeof(timebuf), RFC1123FMT, gmtime(&now));
//timebuf holds the correct format of the time.
```

In client.c, you need to:

1. Parse the <URL> given in the command line.
2. Connect to the server
3. Construct an HTTP request based on the options specified in the command line
4. Send the HTTP request to server
5. Receive an HTTP response
6. Display the response on the screen.

After constructing the http request and before you send it to the server, print it to stdout in the following format:

```
printf("HTTP request =\n%s\nLEN = %d\n", request, strlen(request));
```

where request holds your constructed request.

After getting the response from the server and printing it to stdout, print the following message:

```
printf("\n   Total received response bytes: %d\n",size);
```

where size is the number of characters in the response.

Your client should close connection after getting the file.

Error handling:

1. In any case of a failure in one of the system calls, use perror(<sys_call>) and exit the program (for errors on gethostbyname call herror instead).
2. In any case of wrong command usage, print "Usage: client [-h] [-d <time-interval>] <URL>"

3. In any case of wrong input, print "wrong input" and exit the program. For example, the protocol is not http or the time is in the wrong format.

   Enter new line after each error message.

Useful function:

1. Strchr
2. Strstr
3. strcat
4. Time
5. Strftime
6. gmtime

Compile the client:

gcc –o client client.c

client is the executable file.

What to submit:

You should submit a tar file called ex2.tar with client.c and README. Find README instructions in the course web-site.

Test the client:

You can use the client to connect to any HTTP server. You should try different URLs and options to make sure that the client works correctly.