Assignment 3

HTTP Client-server (Extension of Assignment 2)
Due: February 11, 2015

Extend the code for Assignment 2 by incorporating the following additional features.

1. Each client will maintain a cache (in a specified directory named *Temporary_files* under the current directory from where the client browser is run). When the user gives a GET command at the prompt, the browser should ask the user "Is a cached copy ok (yes/no)?". If the user types in "yes", the page is first searched for in the cache and retrieved from the cache if found (and so no connection is made to the server), otherwise it is retrieved from the URL location as usual (normal execution of GET function in Assignment 2). If the user types in "no", the *cache-control* field is set to *no-cache*, and the page is fetched from the URL as usual (normal execution of GET function in Assignment 2). If the answer is anything other than yes/no, the question is asked again and again until the user enters yes or no.

   Similarly, in a response to a GET request, if the *cache-control* field is not present, the page (along with its URL) is cached in the specified directory. If the *cache-control* field is present and has value *no-cache*, the page is not cached. The server should fill up the field *Last-Modified* in the Entity header with the last modification time of the file sent, and this should be cached along with the page. Thus, every cache entry has a URL, the content of the page, and the *Last-Modified* time for the page (you can encode the URL and the time in the file name itself suitably).

2. An additional REFRESH command is to be supported. Format: `MyBrowser> REFRESH {URL}.` If the URL is not present in the cache, the page is fetched from the server with a GET method as usual. If the URL is present in the cache, a GET request is sent to the server with the following addition - the *last modified* information for the page (available in the cache) is copied in the *If-Modified-Since* tag of the request. The server checks if the value in the *If-Modified-Since* tag sent by the client is earlier than the current last modification time of the page requested; if yes (the page has been changed since the last time it was downloaded by the client), the page is sent as a normal response to a GET command. If not (the copy in the client's cache is the most updated one), the server sends an error code 304 (not modified) to the client with no message body, and the client shows the page from the cache.

3. For every client request, the server first checks the number of active connections at that instant. This is done right at the point the server comes out of the *accept* call (you can keep track of the number of active connections by simply keeping a shared integer that is incremented on each connection established and decremented right before the connection is terminated). If the number of connections (MAX-CONN) is more than a specified constant (say 2, #define it at the beginning), then the server sends an error code 503 (Service unavailable) with the *Retry-After* tag filled with an integer that indicates the

number of seconds to try after. The client should print a message to the user “Server busy: Please retry after *x* seconds” where *x* is the value received in the *Retry-After* tag.

Other functions of the HTTP Client-Server will remain same as in Assignment 2. Look up the time format to put in the *Last-Modified* and *If-Modified-Since* tags from the RFC. You should submit two C files, the *MyBrowser-ext.c* and *MyHTTP-ext.c*.

To test the caching part, the server needs to set the *cache-control* field at least for some files. You can set it randomly or in an on-off fashion (first file sent has *cache-control=no-cache*, second file does not have the field, third has the field and so on).

To test the third part, you can put in sleep in the server threads to delay servicing the requests so that you can try more connections from other terminals and see if you are getting the correct error code.