COMPUTER NETWORKS

PROGRAMMING ASSIGNMENT 3

1 Problem description

Write a simple web server that displays pages from specified directory. Follow preparatory steps below that will allow you to test your implementation:

1. Fetch web pages archive containing 3 web pages and unpack it to directory of your choice.
2. Configure local DNS name resolution by adding following entries to `/etc/hosts` file:

```
127.10.10.1    dom1.abc.pl
127.10.10.2    dom2.abc.pl
127.10.10.3    dom3.abc.pl
127.10.10.4    dom4.abc.pl
```

From now on references to these domains will be directed to your computer (`localhost`).

The server should accept two parameters given on the command line: port number and directory path. The first one is the port number on which the server is listening for incoming connections, and the second is the directory that contains web pages (i.e. fetched ones). The program should handle erroneous input (for instance: a non-existent directory) and should indicate such conditions by reporting a message.

If a user is running web browser on the same computer as the server, opening web page under `http://host-domain:port/page.html` should trigger the contents of `directory/domain-name/page.html` to be displayed.

1.1 Implementation

Of course, you don't have to implement full HTTP protocol. It will suffice for your server to handle only `GET` requests. Just read the first line of such query (i.e. the address of requested path), `Host` and `Connection` fields (see below). In particular you should not support conditional `GET` requests (e.g. with `If-Modified-Since` field).

Your server should send back the following information: response code, `Content-Type` and `Content-Length` fields.

1. Your server should return following response codes (if necessary you can implement some other defined in HTTP standard):
   - `200 OK`: if handling request succeeded;

   - `301 Moved Permanently`: if the browser wants to retrieve an object that is a directory, then it should be redirected to `index.html` page, which is located inside the directory;
   - `403 Forbidden`: if the browser wants to fetch a page from a domain that is not handled by the server;
   - `404 Not Found`: if the browser wants to retrieve a page that does not exist;
   - `501 Not Implemented`: if the browser sends a request other than `GET` or if the data sent is malformed;

   If the response code is different from `200`, your server should put a message into the body of HTML packet. The message represents a simple description of the condition that occurred. After retrieval it is displayed by the browser.

2. `Content-Type` field should be set based upon file extension. Following files should be handled properly `txt`, `html`, `css`, `jpg`, `jpeg`, `png` and `pdf`. Set `application/octet-stream` for other file types.

Your server should not disconnect immediately after a request has been handled, but rather maintain connection with an inactive client for a defined period of time (e.g. 1 second). The exception is when the browser sends `Connection: close` header, in which case you should close the connection immediately after handling the request. On the same connection your server should be able to handle multiple requests (e.g. web page and associated images). Your program may, but does not have to, handle multiple connections simultaneously.

Remember that the web server must not send the contents of any file lying outside of the directory that contains all files for given domain. In addition, your program should be resistant to attacks from malicious users (e.g. sending garbage instead of the correct HTTP request). Your program may display diagnostic messages (e.g. information about the HTTP requests and responses), but they should be concise and clear.

2 Technical remarks

Files The program should be sent to your teacher as a single compressed archive (preferably in *.tar.gz* format). It must contain a single directory and within it:

- Source code in C or C++, i.e. `*.c` and `*.h` files or `*.cpp` and `*.h` files. Each of `*.c` and `*.cpp` source files must contain a comment (at the beginning of file) with following information: first and last name, an ”indeks” number.
- `Makefile` file, that enables the program to be compiled through running `make` command.
- Optionally, `README` file containing any remarks from a student.

The directory **must not** any other files that mentioned above, and especially compiled program (in binary form) and linkable objects `*.o` or any other files not required to build the program.

Compilation The program will be compiled and run in Linux 64-bit environment with fairly recent `GCC` compiler.

In case of C language compilation, your program is allowed to use ISO C99 standard, possibly with GNU extensions (compiler option `-std=c99` or `-std=gnu99`).

Compilation will be performed with `-Wall` and `-W` options. The compiler must not report any warning with regards to your code.

Marcin Bieńkowski
Translation: *Krystian Bacławski*