Programming Assignment 1
Introduction to Socket Programming in Java (or C/C++)

1 Introduction

Berners-Lee and his team are credited for inventing the original Hyper Text Transfer Protocol (HTTP) along with Hyper Text Markup Language (HTML) and the associated technology for a web server and a text-based web browser. The first version of the protocol had only one method, namely GET, which would request a page from a server. The response from the server was always an HTML page. What you're about to do is to reinvent the wheel on the motivation of getting a deep understanding of how HTTP works! In this assignment, you will use sockets to implement a simple web client that communicates with a web server using a restricted subset of HTTP. The main objective of this assignment is to give you hands-on experience with UNIX sockets.

2 Part 1: Multi-threaded Web Server

2.1 Introduction and Background

Please refer to the lectures' slides for the format and the use of HTTP.

3 Specifications

Your web server should accept incoming connection requests. It should then look for the GET request and pick out the name of the requested file. Note that a GET request from a real WWW client may have several lines of optional information following the GET. These optional lines, though, will be terminated by a blank line (i.e., a line containing zero or more spaces, terminated by a '\r\n' (carriage return then newline characters). Your server should first print out the received GET command as well as any optional lines following the GET (and preceding the empty line). The server should then respond with the line, this is a very simple version of the real HTTP reply message:

```
HTTP/1.0 200 OK\r\n
{data, data, ...., data}
```

followed by a blank line (i.e., a string with only blanks, terminated by a ' ° '). The server should then send the contents of the requested document, close the socket created by the accept() function and wait to accept a new connection request. If the document is not found, the server should respond with(as would a real http server) :

```
HTTP/1.0 404 Not Found\r\n
```

3.1 Server Side Pseudo Code

```
while true: do
    Listen for connections
    Accept new connection from incoming client and delegate it to worker thread/process.
    Parse HTTP/1.0 request
    Ensure well-formed request (return error otherwise)
    Determine if target file exists and if permissions are set properly(return error otherwise)
    Transmit contents of file to connect (reads from the file and writes on the socket)
    Close the connection
end while
```

3.2 Notes

- There are different types of HTTP status codes, you're only required to handle 404 and 200.
- You will want to become familiar with the interactions of the following Java Classes: ServerSocket, Socket. For C, we need to use socket(), select(), listen(), accept(), connect().
- Command to run the server:

  ```
  ./my_server port_number
  ```

- You are supposed to handle HTML, TXT and images.

4 Part 2: HTTP Web Client

4.1 Specifications

Your web client must read and parse a series of commands from the standard input. For this assignment, only the GET command is required to be handled. The get command syntax should be as follows:

```
GET file-name host-name (port-number)
```

Note that the port-number is optional. If it is not specified, use the default HTTP port number, 80. In response to a GET operation, the client must open a connection to an HTTP server on the specified host listening on the specified (or default) port-number. It must then transmit a legal HTTP GET request to request the file specified. It must display the file and then store it in the local directory (i.e., the directory from which the client program was run). The client should shut down when reaching the end of file.

4.2 HTTP commands sent by the Client:

This is a real HTTP GET command example:

```
GET /somedir/page.html HTTP/1.0
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
(extra carriage return + line feed)
```

In this assignment and for simplicity, performing a GET operation with a server is done by sending two lines across a socket connection that has been established with the server. The first line is a string containing the key word GET, followed by a file name, followed by the version of HTTP supported by this client. The second line consists only of a newline character. For example, if the client sends the two strings:

```
GET /index.html HTTP/1.0\r\n
\r\n
```

to an http server, the http server will respond with the file /index.html. You don't need host-name or port-number here as the connection is already established.

4.3 Client Side Pseudo code

```
while more GET operation exists do
    Create a TCP connection with the server
    Wait for permission from the server
    Send next GET requests to the server
    Receives data from the server
    Close the connection
end while
```

4.4 Notes

- You can choose between having an input file that contains a number of GET requests, or let the input file contain only one GET operation for a file (Which contains other GET requests) at the server and upon receipt of the file, you parse it and handle the GET operations one by one, and you keep doing this till you receive the last file which is empty.
- Command to run the client:

  ```
  ./my_client server_ip port_number
  ```

- Your client program should use the reliable stream protocol, i.e., TCP.

5 Bonus

5.1 HTTP 1.1

If a web page contains 4 images, a total of five separate connections will be made to the web server to retrieve the html and the four image files. Note that the previous discussion assumes the HTTP/1.0 protocol which is what you will be supporting in this first assignment. Next, add simple HTTP/1.1 support to your web server, consisting of persistent connections and pipelining of client requests to your web browser. You will also need to add some heuristic to your web server to determine when it will close a "persistent" connection. That is, after the results of a single request are returned (e.g., index.html), the server should by default leave the connection open for some period of time, allowing the client to reuse that connection to make subsequent requests. This timeout needs to be configured in the server and ideally should be dynamic based on the number of other active connections the server is currently supporting. That is, if the server is idle, it can afford to leave the connection open for a relatively long period of time. If the server is busy, it may not be able to afford to have an idle connection sitting around (consuming kernel/thread resources) for very long.

5.2 Test your server with a real web browser

Test your server with a web browser of your choice.

6 List of Useful Resources

- Socket Tutorial: `https://docs.oracle.com/javase/tutorial/networking/sockets/`
- KKMultiServerThread.java: `https://docs.oracle.com/javase/tutorial/networking/sockets/examples/KKMultiServerThread.java`
- KKMultiServer.java" `https://docs.oracle.com/javase/tutorial/networking/sockets/examples/KKMultiServe java`

7 Policy

- You should develop your application in Java (or C/C++) programming language, in teams of two.
- You are required to submit external documentation for your program and to comment your code thoroughly and clearly. Your documentation should describe the overall organization of your programs, the major functions and data structures.
- No Late submission is accepted.