Inf-2301: Assignment 1

Submission deadline: 23:59 Friday September 11 2015

1 RTFM

For the Inf-2301 assignments, we are not going to provide you with complete information about protocols and the tools you need to use in order to answer all questions and write your code. You will be expected to nd and read the documentation that is widely available on the Internet or on the machines in the student lab (in the form of manual pages for example). Your fellow students can also be a great source of general information. When the information you have collected is not clear enough or you get stuck in your search for relevant documentation, you can always ask for assistance from the Inf-2301 staff. Until then, the following assistance to RTFM may be of help.

2 Manual pages

If you have never used the **man** command before, it is time to start now. The **man** command on Unix-like systems will give you a manual page for most commands and tools available on the system, for example: **man pwd** gives you an explanation of the **pwd** command. Another source of information (sometimes better than manual pages on GNU/Linux systems) are the so-called info pages. Try for example: **info pwd** for the info page on the **pwd** command. Reading manual pages effectively is something you have to learn, and spending some time 'getting used to them' typically pays itself back later. Reading the following manual pages are a good start: *man*, *apropos*, *whatis*, *whereis*, *locate* and the introduction pages for each manual section.

2.1 Internet Standards

Another important source of information you will need to consult regularly are the actual Internet standards produced by the IETF (The Internet Engineering Task Force) and other organizations. Each distinct version of an Internet standards-related specication is published as part of the "Request for Comments" (RFC) document series. This archival series is the offcial publication channel for Internet standards documents and other publications of the Internet Architecture Board (IAB) and the Internet Engineering Steering Group (IESG), and Internet community. RFCs can be obtained from a number of Internet hosts using anonymous FTP, the World Wide Web, and other Internet document-retrieval systems. If you are interested in the Internet standards process itself or just want a better understanding of the terminology used in them, then RFC 2026[1] may be of interest. Also, the IETF newcomer's presentation[2] or the "Tao of the IETF"[3] give a good idea of the standardization process and the organizations involved.

[1] Search for RFC at http://www.ietf.org/rfc.html (just type in the RFC number)
[2] http://www.ietf.org/proceedings/07dec/slides/newcomer-0/sld1.htm
[3] http://www.ietf.org/tao.html

2.2 Other Sources of Information

Often overlooked nowadays (because you cannot click on them?), but extremely valuable sources of information are libraries. A library is also something you need to learn how to use, so it never hurts to 'just' spend some time in one to get familiar with what treasures can be found there. If you prefer to click on things and stare at computer screens you can turn to a digital library. An excellent example is the digital library[4] of ACM (Association for Computing Machinery). At the welcome page of the ACM Digital Library you will see "University of Tromso" when accessing this resource from computers on campus. Another good source is IEEE Xplore[5]. At the welcome page of IEEE Xplore you should see the message "Access provided by: UNIVERSITY OF TROMSE" when accessing this resource from computers on campus. Be aware that both digital libraries will give you only limited access if your computer is not recognized as a campus computer. This is services that our university library pays for, and access is granted based on IP-addresses.

3 Part A: Web Server

This assignment is about the Hyper Text Transport Protocol (HTTP) or web servers. Web servers are HTTP servers that respond to HTTP requests by returning the appropriate web page, generally represented by a Hypertext Markup Language (HTML) file. The current HTTP protocol is described in a series of RFCs: 7230–7235.

3.1 Implementation details

The first part of the assignment is to implement a simple web server. Since this is a socket programming exercise, we are doing some simplifications for the implementation of the web server. Your implementation should run on localhost and port 8080. Your implementation must support the HTTP request codes: GET, POST. It should also support response codes "OK" and "NOT FOUND". GET should be implemented to open the file at the path from the URL. The start (root) folder is the location of the running server. If the path is not a HTML file, you should try to open the **index.html** at the given path. POST should be implemented to write the input data into the file given in the URL (path). An important detail to be aware of is that you need to handle GET requests for more than just **index.html**, in other words a general approach. This is so that you can verify that your implementation of POST actually works, the testpage.html does this verification, you can also try to handle the **favicon.ico**, but this is optional. Also remember that you should implement a web server from scratch, not use an existing implementation or a Python module implementing most of the web server. Implementations based on **SimpleHTTPServer**, **Flask**, **Django** or similar will not be approved. This means that you need to read up on socket programming in Python using the Socket.py library (or equivalent in other languages).

Also be aware that we expect you to handle content length of anything that is sent from the server. You should read up on the RFC documents mentioned earlier for a closer description of how content-length works in HTTP headers.

3.2 Testing

To test the implementation, a simple HTML file is made avilable in Fronter (see **inf2301-resources-assignment1.zip**). When running your web server, you should be able to view this in a web browser at **http://localhost:8080/**.

[4] http://portal.acm.org/dl.cfm
[5] http://ieeexplore.ieee.org/

4 Part B: REST

In this assignment, we shall use HTTP to implement a RESTful application. REST (Representational State Transfer) is a client server based software architecture. Clients initiate requests and servers respond accordingly. See "Principled Design of the Modern Web Architecture"[6] (Roy T. Fielding and Richard N. Taylor) for more details on REST. Many more (and more compact) introductions to REST are available.

4.1 Implementation details

The application you are to implement is a message server. The server should store all messages in a XML file with the following structure:

```
<?xml version="1.0"?>
<messages>
    <message id="0" value="This is a test message."/>
    <message id="1" value="This is another one."/>
</messages>
```

You have to implement the following requests:

- A GET request to the server should return all messages in the XML format described above.
- A POST request should store the field *message* of the request as a new message, give it a unique id, store the new message with id into the xml file, and return the id back to the client.
- A PUT request with an *id* and a *message* field as payload should update the message with the given id to contain the given message.
- Finally, a DELETE request should delete the message with the given *id* field.

Your implementation should run on localhost and port 8080.

4.2 Testing

To test the implementation a Python script is made available in Fronter (see **inf2301-resources-assignment1.zip**).

5 Report

For part A and B describe the design and implementation of the programs you made. Do not simply copy a lot of content from external sources like RFCs, refer to them instead. Do not include the source code in the text (it should be submitted as separate files). However, the source code should be readable enough, both in structure and the way it is commented, for another programmer to verify its correctness. In other words, your report with source code needs to convince us that your programs work, not execution of the actual programs! Include also a description on how you tested your programs.

For part B you should produce an analytic text tied to your description of part A. The text should not assume that the reader know the detailed content of your text books, and it should refer to external sources used.

[6] http://dl.acm.org/citation.cfm?doid=514183.514185

Include a 'discussion section' in which you note your observations on this assignment, the tools, protocols and libraries you studied, and the documentation you consulted. Maybe you observed something interesting (or just surprising to you). If so, write about it in your report.

Describe what aspects of the solution that caused you implementation headaches, if any. Also report any particular assistance you got from course staff or your fellow students that has been crucial to the fulfillment of the assignment. If an RFC or document was hard to read/interpret for you, make a note of it. You are strongly advised to write documentation/notes along the way.

6 Deliverables

This assignment must be fulfilled by every individual student (no group delivery). You may use your favorite programming language, whatever that may be. Reasonable choices include C, Java, Python and Ruby.

Test your programs on one of the machines in the student lab and make sure that the compilation and execution of your program does not rely on any particular libraries, header files etc. that are only available on your private machine.

Please follow these guidelines for the delivery of your result. Create a directory named after your UiT user name (e.g. abc012). This directory should contain all your deliverables. Assignment documentation and report can be composed in any program, but must be delivered either in adobe acrobat format (pdf) or postscript (ps).

For source code, create a sub directory called *src*. This directory should contain a README file in plain ASCII text, describing how to build (compile) your project, install it (if needed) and how to run it (arguments, etc).

The entire directory tree should be archived in a tar/gzip or zip file. This file should be named as the assignment directory (with the appropriate *.tar.gz* or *.zip* suffix). Finally use Fronter to upload this file to the "Inf-2301 assignment 2" hand-in folder in the Inf-2301 room.