COMP6461 – Fall 2017
Data Communications & Computer Networks
Lab Assignment # 1
Due Date: Friday, October 6, 2017 by 11:59PM

Introduction

In this assignment, you will implement a simple HTTP client application and experiment it in real HTTP Servers (web servers). Before starting on this assignment, it is strongly recommend that you read the provided programming samples and review the associated course materials.

Outline

The following is a summary of the main tasks of the Assignment:

1. Setup your development and testing environment.
2. Study HTTP network protocol specifications.
3. Build your own HTTP client library.
4. Program your HTTP client application (**curl** command).
5. (optional) Implement more HTTP protocol specifications.
6. (optional) Enhance the functionalities of the HTTP client.

Objective

The goal of this Lab is to have your fist steps in implementing network protocol from its technical specifications. Generally, the specifications of network protocol (for example FTP for file transfer and NTP for time synchronization) are provided by standardization organizations like IETF (http://www.ietf.org).

A protocol is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of a physical quantity. These are the rules or standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both. [1]

Network protocols standards are used for various purposes and different OSI - TCP/IP models layers. In the context of this Lab, we focus on implementing as subset HTTP network protocol specifications on top of the **transport layer**, **TCP/UDP** of **OSI**, **TCP/IP** models respectively.

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. [2]

For this purpose, you are requested to implement the basic functionalities of **cURL** command line, the functionalities that are related to HTTP protocol.

cURL is a computer software project providing a library and command-line tool for transferring data using various protocols. The cURL project produces two products, libcurl and cURL. It was first released in 1997. [3]

cURL is an open source command line tool and library for transferring data with URL syntax, supporting DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMTP, SMTPS, Telnet and TFTP. curl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, HTTP/2, cookies, user+password authentication (Basic, Plain, Digest, CRAM-MD5, NTLM, Negotiate and Kerberos), file transfer resume, proxy tunneling and more. [4]

Similarly to the cURL project, the requested implementation in this Lab need to have a network protocol software library and the command line application. Therefore, the main tasks of this Lab are the following:

1. Implementing your own HTTP client library using TCP Socket directly. Only a subset of HTTP protocol features are requested. Specifically, you need to implement GET and HTTP POST requests.
2. Leverage the developed HTTP client library to develop a command-line application similar to the cURL one. However, your version needs only to have a basic functionalities with limited options such as the POST, GET, and verbose option.

Get Started

To get started with the assignment, you will need to setup your development and testing environment based on the chosen programming language. We can use one of the following programming languages: Java, C, Python3, Go, Javascript, and you might want to consider the one that you are most familiar with; since the focus of this lab is on network programming in general and not on coding with a particular programming language.

However, you should notice that the chosen programming language could have a great impact on your progress in the Lab since it may influence your productivity. The later could be increased due to two main reasons: i) the simplicity of language; for example, it might be easier to use Java than C, and Python than Java. ii) The provided network library API could be simpler and more expensive than others for network programming. To this end, we strongly recommend that you choose thoughtfully at the beginning of the Lab. Finally, and for realistic reasons, lab instructors' support could be limited to some programming languages, and support is largely given for Python or Java. So, while we do not want to limit you to use a particular programming language, you should consider these different factors.

We mean by the development environment, the compiler/interpreter of the chosen programming language, the selected programing libraries (socket library), since some programming languages have multiple ones, etc. In addition, we urge you to use a code version control system. In this lab, we use Git for this purpose. Last but not least, you should consider the operating system and the text editor/IDE of your choice. The environment could be setup on a physical machine (laptop), virtual machine (VritualBox), or container machine (Docker). However, the last two choices provide more flexibility in terms of portability and the backup simplicity.

Afterward, you are requested to test the provided samples, *EchoClient* and *EchoServer* to check the development environment. Again, it is recommended that you use one of the following programming languages: Java, C, Python3, Go, Javascript. In the case that you choose a different programming language than those recommended ones, you have to ensure a proper/working development environment by programming and testing the samples yourself; in other words, support from lab instructors may be minimal.

The setup of the development environment is the first step of this first lab assignment, and you only need do it once, then you can use it for the following lab assignments.

Software and Hardware Requirements

The assignment requires a typical machine for the programming and the execution of the produced application. Meaning, the computer should have at least Intel Core 2 CPU, 1GB RAM, and 10GB free space on the disk. However, this could change depending on your development environment choices. These specifications should be higher when choosing to use Virtual Machine systems, which we recommend, such as VirtualBox [virtualbox.org] since running virtualization requires more resources.

Software requirements vary based on the selected programming language. For Python, Java, Go, and javascript, you are free to select the operating system from any of the major ones, Windows, MacOS, and GNU/Linux. However, for the case of C language, you need to use Unix-like operating system and Berkeley socket.

You can setup the development environment according to your preferences, like IDE or editor, version control system, network tools, etc. However, we highly recommend using Git as version control system, Wireshark as network tool.

Git is a version control system that is used for software development and other version control tasks. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows. Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. [5]

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues. [6]

When you finish this Setup you should check with the Lab Instructor

---

Study HTTP Protocol

In this step, you are requested to have an overview of HTTP protocol. Specifically, you should have a detailed understanding of The HTTP GET request and POST response. HTTP protocol has multiple versions, 1.0, 1.1, 2.0, which you can consider them references for you implementation (Only versions >= 1.0). However, we urge you to use HTTP version 1.0 due to its simplicity and ease to implement.

You can find the complete specifications of HTTP protocol version 1.0 at the following link HTTP. However, we believe that the best way to learn is by doing; meaning, experimenting sending requests and receiving responses using the **telnet** command line.

Telnet is an application layer protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP). Telnet was developed in 1969 beginning with RFC 15, extended in RFC 854, and standardized as Internet Engineering Task Force (IETF) Internet Standard STD 8, one of the first Internet standards. [8]

Following example depict how to **telnet** command line to send a simple HTTP GET Request to `http:/`/`httpbin.org/status/418` webpage.

```
> telnet httpbin.org 80

GET /status/418 HTTP/1.0
Host: httpbin.org

HTTP/1.1 418 I'M A TEAPOT
Server: nginx
Date: Sat, 29 Jul 2017 21:58:24 GMT
Content-Length: 135
Connection: close
Access-Control-Allow-Origin: *
x-more-info: http://tools.ietf.org/html/rfc2324
Access-Control-Allow-Credentials: true


    -=[ teapot ]=-

       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
Connection closed by foreign host.
```

To reproduce the example, your should follow the following steps:

1. Launch your terminal/command console. Most operating systems have the **telnet** command line installed. If this is not the case on your version, you need to install it.
2. In the terminal type `telnet httpbin.org 80`, where httpbin.org is the HTTP server and '80' is the standard HTTP TCP port.
3. Once **telnet** connected, type `GET /status/418 HTTP/1.0`, then press Enter in the keyboard.
4. Afterward, type `Host: httpbin.org` then two times Enter to send the request to **httpbin.org** HTTP Server.

You should receive the response depicted in the output sample. Similarly, you can explore HTTP protocol operations without the need to program.

HTTP Client Library Implementation

After getting a deeper understanding of the flow of HTTP GET and POST operations, you are requested to implement your HTTP client using TCP Sockets. The programming library implements only a small subset of HTTP specifications. In other words, we expect that your HTTP library supports the following features:

1. GET operation
2. POST operation
3. Query parameters
4. Request headers
5. Body of the request

You can refer to HTTP protocol reference for more information about these features. You can start your implementation by leveraging the provided samples for HTTP-ECHO and HTTP-TIME protocols at the example directory.

To this end, you should build your testing demos for the HTTP client library in order to demonstrate it.

When you finish the demos you should check with the Lab Instructor

---

cURL-like Command Line Implementation

At this stage, your ready to build a simple HTTP client using your library. In this task, you are requested to implement cURL command line with basic functionalities.

For this purpose, you need to experiment with cURL command line in your preferred terminal in order to know its basic options. You should notice that our focus is on the options related to HTTP protocol since cURL support many network protocols as described earlier.

The implemented client should be named **httpc** (the name of the produced executable). The following presents the options of your final command line.

```
httpc (get|post) [-v] (-h "k:v")* [-d inline-data] [-f file] URL
```

In the following, we describe the purpose of the expected **httpc** command options:

1. Option **-v** enables a verbose output from the command-line. Verbosity could be useful for testing and debugging stages where you need more information to do so. You define the format of the output. However, You are expected to print all the status, and its headers, then the contents of the response.

2. **URL** determines the targeted HTTP server. It could contain parameters of the HTTP operation. For example, the URL 'https://www.google.ca/?q=hello+world' includes the parameter **q** with "hello world" value.
3. To pass the headers value to your HTTP operation, you could use **-h** option. The latter means setting the header of the request in the format "*key*: *value*." Notice that; you can have multiple headers by having the **-h** option before each header parameter.
4. **-d** gives the user the possibility to associate the body of the HTTP Request with the inline data, meaning a set of characters for standard input.
5. Similarly to **-d**, **-f** associate the body of the HTTP Request with the data from a given file.
6. **get**/**post** options are used to execute GET/POST requests respectively. **post** should have either **-d** or **-f** but not both. However, **get** option should not used with the options **-d** or **-f**.

Detailed Usage

General

```
httpc help
```

```
httpc is a curl-like application but supports HTTP protocol only.
Usage:
    httpc command [arguments]
The commands are:
    get     executes a HTTP GET request and prints the response.
    post    executes a HTTP POST request and prints the response.
    help    prints this screen.

Use "httpc help [command]" for more information about a command.
```

Get Usage

```
httpc help get
```

```
usage: httpc get [-v] [-h key:value] URL

Get executes a HTTP GET request for a given URL.

   -v             Prints the detail of the response such as protocol, status,
and headers.
   -h key:value   Associates headers to HTTP Request with the format
'key:value'.
```

Post Usage

```
httpc help post
```

```
usage: httpc post [-v] [-h key:value] [-d inline-data] [-f file] URL

Post executes a HTTP POST request for a given URL with inline data or from
file.

   -v             Prints the detail of the response such as protocol, status,
and headers.
   -h key:value   Associates headers to HTTP Request with the format
'key:value'.
   -d string      Associates an inline data to the body HTTP POST request.
   -f file        Associates the content of a file to the body HTTP POST
request.

Either [-d] or [-f] can be used but not both.
```

Examples

Get with query parameters

```
httpc get 'http://httpbin.org/get?course=networking&assignment=1'
```

Output:

```
{
  "args": {
    "assignment": "1",
    "course": "networking"
  },
  "headers": {
    "Host": "httpbin.org",
    "User-Agent": "Concordia-HTTP/1.0"
  },
  "url": "http://httpbin.org/get?course=networking&assignment=1"
}
```

Get with verbose option

```
httpc get -v 'http://httpbin.org/get?course=networking&assignment=1'
```

Output:

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 1 Sep 2017 14:52:12 GMT
Content-Type: application/json
Content-Length: 255
Connection: close
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

```
{
  "args": {
    "assignment": "1",
    "course": "networking"
  },
  "headers": {
    "Host": "httpbin.org",
    "User-Agent": "Concordia-HTTP/1.0"
  },
  "url": "http://httpbin.org/get?course=networking&assignment=1"
}
```


Post with inline data

```
httpc post -h Content-Type:application/json --d '{"Assignment": 1}'
http://httpbin.org/post
```

Output:


```
{
  "args": {},
  "data": "{\"Assignment\": 1}",
  "files": {},
  "form": {},
  "headers": {
    "Content-Length": "17",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "Concordia-HTTP/1.0"
  },
  "json": {
    "Assignment": 1
  },
  "url": "http://httpbin.org/post"
}
```


When you finish your cURL command-line you should check with the Lab Instructor

---

Optional Tasks (Bonus Marks)

If you have successfully completed the material above, congratulations! The reason behind such congratulations is that you now understand the implementation of an HTTP client and network protocols in general. For the rest of this lab exercise, we have included the following optional tasks. These optional tasks will help you gain a deeper understanding of the material, and if you can do so, we encourage you to complete them as well. Bonus marks will be given for that.

Enhance Your HTTP Client library

In the current HTTP library, you already implemented the necessary HTTP specifications, GET and POST. In this optional task, you need to implement one new specification of HTTP protocol that is related to the client side. For example, you could develop one of the Redirection specifications. The latter allow your HTTP client to follow the first request with another one to new URL if the client receives a redirection code (numbers starts with 3xx). This option is useful when the HTTP client deal with temporally or parental moved URLs. Notice, you are free to choose which HTTP specification to implement in your HTTP library. After selecting the specification, you should consult with the Lab Instructor before starting their implementations.

Update The cURL Command line

Accordingly, you could add the newly implemented HTTP specifications in your HTTP library to the **httpc** command line. To do that, you need to create a new option that allows the user the access the newly implemented specification. In addition, you are requested to add the option **–o filename**, which allow the HTTP client to write the body of the response to the specified file instead of the console. For example, the following will write the response to **hello.txt**:

```
httpc -v 'http://httpbin.org/get?course=networking&assignment=1' -o hello.txt
```

Testing, Submission and Grading

Important Note: You can this assignment individually or in a group of **at most 2 members** (i.e. you and another student). No extra marks or any special considerations will be given for working individually.

Testing

You should be able to test your application with any HTTP server. The httpbin.org provides an excellent service for you to check your command-line. You can write a small utility that calls both your **httpc** and cURL against the same URL with the same parameters and then compare the responses.

Deliverable

1) Create one zip file, containing the necessary source-code files (.java, .c, etc.)

You must name your file using the following convention:

If the work is done by 1 student: Your file should be called A#_studentID, where # is the number of the assignment. studentID is your student ID number.

If the work is done by 2 students: The zip file should be called A#_studentID1_studentID2, where # is the number of the assignment. studentID1 and studentID2 are the student ID numbers of each of the group members.

2) Assignments must be submitted in the right folder of the assignments. Upload your zip file at the URL:

https://fis.encs.concordia.ca/eas/ as **Programming Assignment 1**. Assignments uploaded to an incorrect folder will not be marked and result in a **zero mark**. **No resubmissions will be allowed.**

Demo

A demo is needed for this assignment and your lab instructors will communicate the available demo times to you, where you **must** register a time-slot for the demo, and you must prepare your assignment and be ready to demo at the start of your time-slot. If the assignment is done by 2 members, then **both members must** be present for the demo. During your presentation, you are expected to demo the functionality of the application, explain some parts of your implementation, and answer any questions that the lab instructor may ask in relation to the assignment and your work. Different marks may be assigned to the two members of the team if needed. **Demos are mandatory. Failure to demo your assignment will entail a mark of zero for the assignment.**

Grading Policy (10 Marks)

1. HTTP Library: total of **7** marks
   - Get request: 3 marks
   - Header: 1 mark
   - Post request (eg. with body): 2 marks
   - Response (eg. parse status, code, header, and body): 1 mark
2. Curl-like app: total of **3** marks
   - Get command: 0.5 marks
   - Post command: 0.5 marks
   - Verbose: 0.5 marks
   - Header: 0.5 marks
   - Inline : 0.5 marks
   - File: 0.5 marks
3. Optional tasks: total of **2** bonus marks
   - Supports redirect: 1.5 marks
   - Supports –o option: 0.5 marks

References

[1] Communications Protocol. https://en.wikipedia.org/wiki/Communications_protocol.

[2] Hypertext Transfer Protocol. https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[3] cURL: https://en.wikipedia.org/wiki/CURL.

[4] cURL: https://curl.haxx.se.

[5] Git: https://en.wikipedia.org/wiki/Git.

[6] Wireshark: https://en.wikipedia.org/wiki/Wireshark.

[7] Telnet: https://en.wikipedia.org/wiki/Telnet.