CSE 150

Final Project

Code Submission Early Bird: Monday, May 23, 2022
Code Submission Regular submissions: Sunday, May 29, 2022

Purpose

The purpose of this project is to gain familiarity with the HTTP protocol and the socket interface in Python. The functionality of this program is similar to Curl. You will develop a program to request and download web objects from any public web server. The user enters a URL as a command-line argument, and the program in turn issues an HTTP GET request for the object. Upon successful retrieval the object is written to a file. If an error or redirection is returned from the web server, the HTTP response message is interpreted and displayed in the terminal.

1. Requirements and Guidelines

- Python3 and VM: The program must be written in Python3 and run on the Mininet VM. Code that is not developed on your Mininet VM might not be compatible with our test scripts.
- Program name: <CruzID>MyCurl.py where CruzID is your UCSC email username
- Allowed libraries: socket, argparse, sys

  Your program MUST assemble the HTTP GET request and send the corresponding bytes through the socket interface.

  > For this project, you may not use any **network library** other than Python's `socket`, an interface for the low-level socket communication. High-level APIs such as `http` and `urllib` are *not* allowed.

- Command-line Arguments: Your program must accept a full URL, specified either with the domain name or the IP Address, entered on the command line.

  e.g., http://www.foobar.com or http://128.64.23.10

  - When the full URL includes the IP address, the hostname must also be given as a second command-line argument. (This fulfills an HTTP 1.1 requirement that GET requests contain the "Host: value" header. This is important!)

  - Destination port numbers can be appended to the URL with a ":".

    Your application **must accept** and issue an HTTP GET request for any given port number. If an exception occurs, your code should catch it.

    If no port numbers are given, then the HTTP request is by default directed to port 80.

1. Example command line (1 argument)

   CruzIDMyCurl.py http://www.google.com:80

   - Indicates full URL containing server hostname and destination port 80
   - Equivalent Curl: ***curl*** http://www.google.com:80
   - Equivalent HTTP request (using C string notation \r and \n for CR and LF, respectively) that your program must generate:

     GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n

2. Example command line (2 arguments)

   CruzIDMyCurl.py http://93.184.216.34:80/foo.html www.example.com

   - Indicates full URL containing IP address of server to send GET request, destination port 80 and the hostname of the web server
   - Equivalent Curl:

     curl http://93.184.216.34:80/foo.html --header 'Host:www.example.com'

   - Equivalent HTTP request (using C string notation for CR and LF) that your program must generate:

     GET /foo.html HTTP/1.1\r\nHost: www.example.com\r\n\r\n

     Socket would be opened with 93.184.216.34:80

- Document Size: The client should be able to work with documents of any size, as long as the computer has enough free space.
- Exceptions:
  - Chunk Encoding: Sometimes the requested object is returned with chunk encoding (https://en.wikipedia.org/wiki/Chunked_transfer_encoding). Your program is not required to support chunk encoding and you can disregard the downloaded object, but an error message must be printed to the terminal stating chunk encoding is not supported and the URL should be entered into log.csv (see below).

  - HTTPS: URLs with HTTPS scheme should be rejected and an error message printed to the terminal stating HTTPS is not supported

- TCP Errors: Make sure to catch TCP errors, such as those received when HTTP requests are issued to servers that don't exist, when a connection is refused to port 443 or when any port is specified and there is no server listening on the destination port. All of these errors should be displayed in a message to the terminal (stderr).

- Closing your program: Your program must close gracefully – make sure to close any open sockets before exiting the program.

2. Program Output:

Your **goal** is to match the output of Curl - i.e., whatever Curl returns, your program interprets the HTTP Responses similarly.

Your program will write to 2 files –

HTTPoutput.html: A successfully retrieved web object is written to this file.

Log.csv: After issuing an HTTP request, the server response is captured in this CSV file. The column names should be on the first line. sourceIP, destination IP, source port, destination port is the 4-tuple. Source is considered to be the Client. The format is:

```
Successful or Unsuccessful, Requested URL ,  Hostname, source IP, destination IP, source port, destination port, Server Response line (including code and phrase)
```

Successful retrieval: If a requested web object is returned in the body of the server's HTTP Response, your program will do the following: 1) Write the received web object to HTTPoutput.html. (Note: Only the requested web object should be written to the file. This file should contain HTML that can be displayed by your browser). 2) print a message to the terminal with the word "Success" followed by the requested URL and the HTTP Response status line and 3) append an entry into log.csv

Unsuccessful retrieval: If a web object is not successfully downloaded, the program does the following: 1) print a message to the terminal with the word "Unsuccessful" followed by the requested URL and the the HTTP Response status line and 2) append an entry in log.csv

3. Getting Started

1. Read Section 2.7 of the textbook Computer Networks: A Top-Down Approach on sockets
2. Take a look at the format of HTTP Request messages and Server responses (class notes, text book and below)
3. Practice with Curl and look at the corresponding steps in Wireshark. Your code needs to mimic these steps.
4. Read through the Useful Resources section (below)
5. Check out the list of plain text website for testing (below)

3. General Strategies

- Get the program working with port 80 and a simple working URL. Work with URLs that respond as expected, then focus on errors. See below for a list of plain text HTTP web servers.
- If the requested URL is downloaded successfully by Curl (think about which HTTP status code indicates this), but not by your program, look at Wireshark. Wireshark displays properly formatted HTTP messages (requests and responses).
- If the requested URL generates an error message, verify the same error is displayed by Curl.
- Curl -I url returns the HTTP headers only, which can be helpful for debugging
- Curl is a tool for you to use -- trouble forming URL in your program? Try it out in Curl!

4. Testing

- A test harness is available here [Sign in with your UCSC Google account]. The tests are *not* exhaustive but cover many common cases. The harness will not run on the VM due to Python3.6+ dependency (sorry!).
    -

```
usage: test.py [-h] [-v] path

positional arguments:
  path           path to program to test

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  enable verbose output
```

- Make sure your requests can be directed to all user-specified destination ports. If there is an error, your program should log and report the error (stderr). Possible errors are:
    - curl: Connection reset by peer
    - curl: Chunked transfer encoding not supported
    - curl: HTTPS not supported
    - curl: Could not resolve host
- When a requested web object is successfully downloaded and saved, view it in your browser. Is it what you expected? Validate your output based on what is returned by Curl for the same URL and port. Compare the HTTPout.html with Curl and make sure they are the same.
- You are free to output any needed debug messages to the terminal (stdout).

- Make sure to check for errors such as malformed URLs, objects not found, HTTPS etc. Here are a few example URLs that will generate these kinds of errors:

  http://www.example.com/anyname.html
  https://www.google.com
  http://www.google.com/somedoc1.html:80
  http://www.google.com:443 (Note: your program should issue this request and not simply filter it out.)

5. Deliverables

1. **README.txt**: A readme file explaining your submission.
2. **<CruzID>MyCurl.py** Your Python code where CruzID is your UCSC email username
3. **Log.csv:** Include a line for each of the URLs you have tested. Make sure you have tested:
   a. 2 successful downloads
   b. 2 unsuccessful downloads
   c. 1 redirect (can be included in the plain text HTTP sites listed below)
   d. a request to port 443, e.g., http://www.google.com:443

4. **Discussion.pdf**:

   This documentation describes how to use your application as well as any shortcomings it might have.

   For each of the test URLs in log.csv:

   - state whether the downloaded object or response was successful or not and if the output matches Curl
   - include screenshots that show the Wireshark capture of the HTTP message exchange. This includes:
     - The HTTP Request (method plus headers)
     - The HTTP Response (status line and headers)

5. **Questions.pdf**

   1. How did you decide which type of socket to use? Why?
   2. How did you choose the destination ports?
   3. What error handling cases did you implement?
   4. How does your program terminate? What happens to the TCP connection?
   5. For the unsuccessful URLs, why were they unsuccessful?
   6. What happens if you try to access a site using HTTPS?

6. Grading

Project Demo: Each student will meet with their TA during an assigned time slot. You will be asked to run your program and discuss your code design. The demo is **mandatory and is the only way to receive a grade for the project.** Missing the demo will result in a score = 0. Once you sign up for a demo slot this is your commitment and it cannot be missed without prior rescheduling and agreement with your TA.

Rubric:

Each submission will be tested to make sure it works properly and handles errors. Grades are allocated using the following guideline:

Functionality including error handling: 50%

(preprocessing tests performed prior to demo)

Questions and log.csv:	5%
Discussion.pdf:	5%
Live TA Demo:	40%

Deadlines:

Please make sure to submit on time. **There is no late code accepted**.

Early Bird: Submit code by Early Bird submission date for an extra 10 points. Demo is during Week 9.

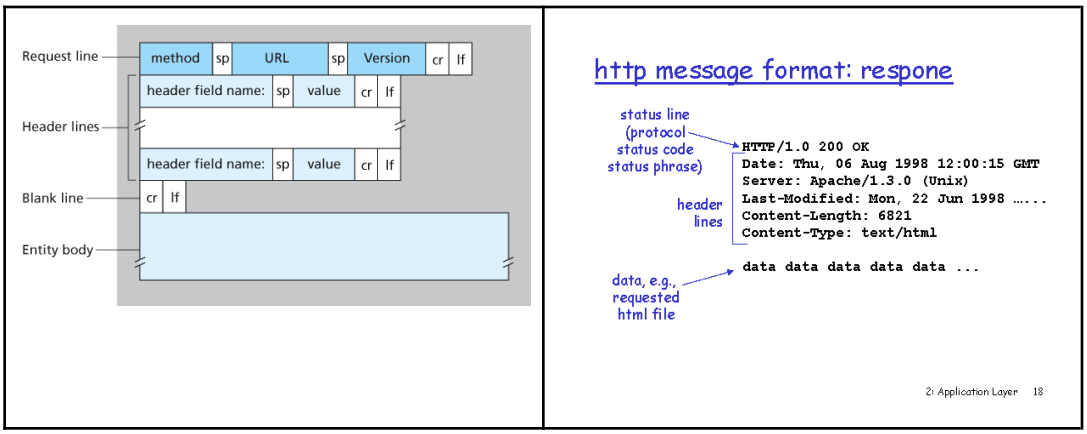
Regular Submission: Demo is during Week 10

7.Honor Code

All the code must be developed independently. All the work must be your own - please remember to cite any sources in your Discussion.pdf document. MOSS will be run on all submissions and any instances of plagiarism will result in a score of 0.

8.Useful Resources

- Computer Networks: A Top Down Approach:
    - Chapter 2 HTTP
    - Chapter 2 Section 7 on TCP sockets
- Lecture material on HTTP and steps for downloading web pages
- HTTP message format - Request and Response:



- Python sockets: https://docs.python.org/3/library/socket.html
- HTTP: HTTP requests:

  https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html • HTTP

  responses: https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html
- Exception Handling in Python
    - https://www.programiz.com/python-programming/exception-handling

- Plain text HTTP websites for testing:

  – http://wwwexample.com/index.html
  – http://www.google.com/index.html
  – http://neverssl.com/
  – http://pudim.com.br/
  – http://www.softwareqatest.com/
  – http://www.testingmcafeesites.com/
  – https://datatracker.ietf.org/doc/html/rfc2616

FAQ

- **Do I have to use the VM for this lab?**
  - Yes. Your program will be evaluated on the VM so, even if you choose to develop on your host machine, please verify your program on the VM for Python3.4 compatibility. Some syscalls may behave differently on your host machine as well.
- **What is the expected result when the IP address doesn't correspond to a real server?**
  - The connection could be refused or failed, the program should terminate gracefully.
- **Where should the response from a server be printed when there is a 404, 403, or any other status code?** Should it go into log.csv or just write a message to the terminal?
  - The status line should be displayed in the terminal and an entry should be appended to log.csv –the same way as if the response had a 200 code.
- **What are "unsuccessful downloads"?**
  - Could be chunked encoding or any other kind of error from the server
- **What if the Transfer-Encoding header is returned with a value of "chunked" in the server's response?**
  - The client should print an error message to the terminal that chunked encoding is not supported for this program and append an entry to log.csv
- https://en.wikipedia.org/wiki/Chunked_transfer_encoding
- **What libraries are allowed?**
  - socket, argparse, sys
- **Should I use python2 or python3?**
  - Python3
- **My program is freezing up and never finishes reading from the socket. What is the problem?**
  - Reading from a socket is a blocking call and your program will hang inside of that function until the socket is closed (timeout) or data is received on the socket. So if the socket finished sending all of its data and you try to receive from the socket another time, you can get stuck inside.
- **Should our program download the "404 not found" object?**
  - Yes, always download everything after the HTTP header.
- **Should our program follow a redirect message**
  - No, you should just download the object if one is returned and log the HTTP response code
- **What constitutes a successful retrieval?**
  - Anytime a web object is requested from the server and returned in the body of the server's HTTP response
- **How should we go about handling when port 443 is specified?** Should we manually parse the input and throw an error or should we have a timeout due to the empty reply after connecting to the socket?
  - You cannot just assume that because it is port 443, there will be an error. You need to catch the exception from the socket. You can look at the documentation for socket.recv() to see the exceptions/error codes.

- **What if I cannot attend my demo?**
    - The completed demo to your TA is the only way to receive credit for the project.