Programming Assignment 1: Socket Programming

Due Feb 15th, 2019

In this assignment, you will familiarize yourself with TCP socket programming and the design of distributed applications. The assignment consists of two parts.

Part I: A Simple Web Server

In this part, you will develop a simple Web server in Python 3 that is capable of retrieving *static* HTML files that contains links to images using HTTP 1.1. Specifically, your Web server will,

(i) bind to and listen a port specified as a command line argument
(ii) create a connection socket when contacted by a client (browser);
(iii) receive the HTTP request from this connection;
(iv) parse the request to determine the specific file and file type being requested;
(v) get the requested file from the server's file system;
(vi) create an HTTP response message consisting of the requested file preceded by proper header lines. Specifically, the HTTP response code, Content-size Content-type fields should be set properly.
(vii) send the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in your server, your server should return a "404 Not Found" error message.

In the PA1 folder, you are provided

- Project description file name description.pdf – this document
- A skeleton code named WebServer.py. You need to fill in the portions marked with "YOUR CODE" in the file. [PLEASE DO NOT CHANGE THE NAME OF THIS FILE].
- index.html and hello_world.jpg files for testing. You are encouraged to develop your own test cases.

For testing, you can run your code from command line, e.g., "python WebServer #PORT_NUMBER" or from an IDE, where #PORT_NUMBER should be within the range of 1024 – 65535 unless you have root permission to your system.

NOTE:
1. The port number of an improperly closed server cannot be immediately reused. You can either wait for a timeout or use a different port number for testing.
2. The skeleton code provided uses multi-threading to allow multiple clients to connect to the server at the same time. Though not crucial to this assignment, you are encouraged to review multithread programming in Python (see the 3rd link in the reference)

Reference:
- Socket programming in Python https://realpython.com/python-sockets/
- Python socket interface https://docs.python.org/2/library/socket.html
- Multithreading in Python https://docs.python.org/3/library/threading.html
- Lecture slides

Part II Design of a peer-to-peer (p2p) file sharing application

Now, consider you are hired by bestcompany.com to develop a p2p file sharing application that synchronize files among peers. It has the flavor of Dropbox but instead of storing and serving files using servers in the cloud, we utilize a peer-to-peer approach. Recall that the key difference between the client-server architecture and the peer-to-peer architecture is that in the later, the end host acts both as a server (e.g., to serve file transfer requests) and as a client (e.g., to request a file). One challenge in peer-to-peer applications is that peers do not initially know each other's presence. As such, a server process, called Tracker is needed to facilitate the "discovery" of peers. For simplicity, we only consider sharing of files in a flat directory.

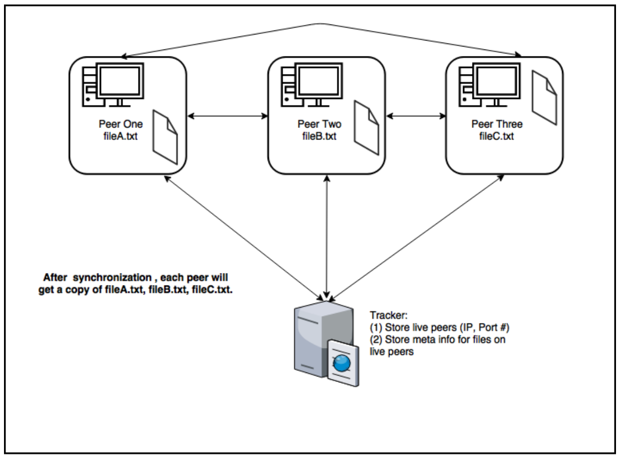
Peer One
fileA.txt
Peer Two
fileB.txt
Peer Three
fileC.txt
After synchronization , each peer will get a copy of fileA.txt, fileB.txt, fileC.txt.
Tracker:
(1) Store live peers (IP, Port #)
(2) Store meta info for files on live peers

Figure 1 Application Architecture

The figure above illustrates the architecture of the application. In the example, "Tracker" traces the live peer nodes and store only the meta info of files (NOT THE ACTUAL FILES!) on 3 peers. In this example, initially, each peer node has one local file (fileA.txt, fileB.txt, fileC.txt). After they connect to the tracker and synchronize with other peers, each peer will eventually have all three files locally.

Answer the following questions regarding a possible design of the software:

1. What kind of information Tracker should maintain for live peers and for the files on each live peer?
2. How can Tracker know if a peer is alive or not?
3. How to ensure Tracker have the most up-to-date information of the files on each live peer, in event of file deletion, modification and addition?
4. For peers to know any change on other peers, either push-based (i.e., Tracker can notify all peers) or pull-based approach (i.e., a peer can pull the information from Tracker periodically) between Tracker and peers can be adopted. Discuss the pro & con of both approaches.
5. How many sockets Tracker should create? Why?
6. How many sockets a peer should create? Why?
7. Assume files are uniquely identified by their file names, and peers and Trackers have common clocks (e.g., through Network Time Protocol). Initially, different peers may have different versions of the same file. How to ensure only the most up-to-date version is synchronized among peers?

Submission: Commit your implementation in WebServer.py and answers to Part II in a PDF file named PA1_Part2.pdf under Gitlab PA1 folder.