Anton Krotenok (NETID: ak1847)

Kevin George (NETID: kmg328)

Brief Explanation of The Project

**In this project we implement four programs (client.py, ls.py, ts1.py, ts.py) to mimic the functionality of a load balancer being used across DNS servers. In theory, input is transferred in the following way: client.py —> ls.py —> (ts1.py OR ts2.py). The output moves in the direct opposite direction (the programs are called in this same order). Depending on if the client is able to receive a valid result from the DNS search, a final document titled RESULT.txt is generated to hold all the final results. A non-blocking socket function, via select.select() is used to prevent either ts1.py or ts2.py from “hanging” or stalling the rest of the processes. This means at most one TS will respond to any query from ls.py. It is also possible that neither TS has a mapped to for the queried domain name. The assignment will be tested with local and remote socket connections.**

1. Team details: Clearly state the names and netids of your team members (there are 2 of you).

**Anton Krotenok (NETID: ak1847)**

**Kevin George (NETID: kmg328)**

2. Collaboration: Who did you collaborate with on this project? What resources and references did you consult? Please also specify on what aspect of the project you collaborated or consulted.

**Both partners utilized past knowledge from classes including Computer Architecture, Software Methodology, and Principles of Information and Data Management. To better understand the functionality of select.select(), we looked for documentation on websites including “Stack Overflow” & “docs.python” .**

3. Discuss how you implemented the LS functionality that tracks which TS responded to a

given query or timing out if neither TS responded. Please be clear and specific.

**The ls.py program begins by establishing the proper socket bindings and connections. All data meant to be transferred between sockets occurs within the main while() loop. It is here that the select.select() function is applied and 5 seconds is given to wait for data to become available. The “inputs” list stores the sockets while the “for” loop iterates through these same objects inside the “readers” list. Data is retrieved (from TS) via the .recv() function and decoding using the UTF-8 encoding. The client socket (“csocketid”) has a .send() method called on it to transfer the information to client.py. “inputs.remove(r)” removes the socket that just had information available.**

**In the case that neither ts1.py or ts2.py have any information to share, the “readers” list will be empty. The variable “d” is also initialized with a “TIME OUT” message and is returned as part of the final message. As explained in more detail for Question #5, we added “if not(readers)” to check against timing out if neither TS responded & a boolean flag conditional to break the loop if data is indeed found.**

4. Is there any portion of your code that does not work as required in the description above?

Please explain.

**There is no portion of our code that does not work as required in the description above.**

5. Did you encounter any difficulties? If so, explain.

**The largest difficulty we had was properly implementing ls.py with the select.select() function. There were two errors we had to solve relating to searches that failed to exist in either TS file: (1.) The project would stop running (program will hang indefinitely) at the existence of the first “...TIMED OUT” response. (2.) The project would return and print a ‘...TIMED OUT” response in conjunction with some valid return from the search.**

**To fix these issues we did the following: (1.) Make sure that ls.py has the proper if conditional to check for the condition in which neither ts1.py or ts2.py have a valid result to return back. By writing “if not (readers)”, we made sure that the client receives some response if select.select() timed out without any incoming data from the monitoring sockets. (2.) Rewrite ls.py and make sure that there is a boolean flag paired with a “if({boolean}): break” conditional in the case that either ts1.py or ts2.py return some valid search output. This helps to prevent select.select() from returning the results of two sockets at the same time.**

6. What did you learn from working on this project? Add any interesting observations not

otherwise covered in the questions above. Be specific and technical in your response.

**This project helped Anton and Kevin mimic the implementation of a load balancer across DNS servers. More specifically, each partner spent extensive time researching the select.select() non-blocking call and trying to learn its advantages against the alternative “manual” solutions that utilize a timing delay and nested try-catch loop. It was important to map a timeline of what functions get called at what time, especially when trying to understand how the different sockets communicate with one another (ex. What are “.bind()”, “.connect()”, “.recv()”, and “.recv()” doing in relation to the other files.) Furthermore, we had the observation that ls.py was the most difficult to implement because it acted both as a client (to ts1.py & ts2.py) and server (to client.py). Issues with port numbers being already in use were present but resolved quickly.**