CSE 5299 – Fall 2024 Programming Assignment #1
Ariannah Black. (October 22, 2024)

UDP Multithreaded Client/Server Design Document

Note: The code and (in-line) documentation for this project can also be found at the following GitHub link: https://github.com/ariblack17/simple-server.

Description

The Server: The server opens a serverSocket on a non-reserved port (13000), then waits for wildcard queries from clients. Each client is handled on a separate thread (up to 5 threads, supporting up to 5 clients in parallel), and each client is able to send multiple queries to the server for multiple responses.

The wildcard query supported in this server follows the same format as in the TCP server.

The Client: The client opens a clientSocket selected by the operating system, sends a wildcard query request to the server, then receives and unpacks a response from the server. This process of sending and receiving messages repeats until the client terminates communications.

Application Protocol: The protocol for this UDP server is the same as in the TCP server, except the Connection-type header in client requests has been renamed to Request-type header.

Trade Offs

The main trade off for the UDP multithreaded server matches the *Program Structure* trade off listed in the TCP document. Developing an object-oriented design in favor of the selected design would have resulted in a different (but practically functionally-identical) set of code.

Extensions

The main extensions for this UDP multithreaded server match the *Improved Structure and Optimization* and *Generating Better "UI" for the Client* extensions listed in the TCP document.

Identifying Source/Destination Addresses: A known requirement for UDP implementation in client/server design is the accession of packet source and destinations in order to correctly route application data to processes. While this is already taken care of

(from the *socket* library in Python), it may be helpful to developers (and code-testers) to be able to read packet sources and destinations within the application protocol headers themselves.

Test Cases

Structure: The following Client Queries section follows the same structure as the TCP test cases – first (in bold/italics) is a line containing the client queries sent to the server for the given test case. Each query is separated by |. Below the query line is a brief description of the test cases' functionalities, and below this descriptor a screenshot showing the programs' outputs for the given queries.

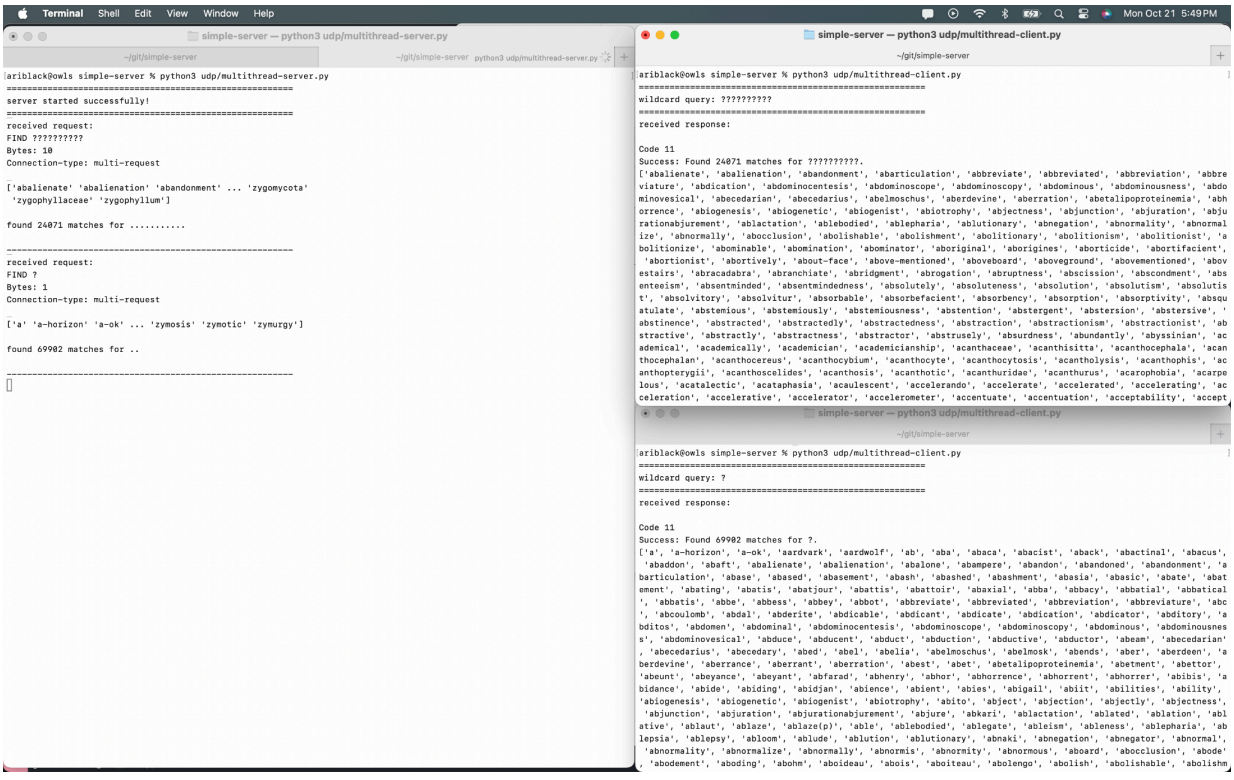
The left half of the screenshot contains a terminal running the server process, and the right half contains two separate terminals running two client processes.

Note that screenshots showing proof of functionality and testing for the basic client/server programs can be found in the GitHub repository linked at the start of this document.

Client Queries:

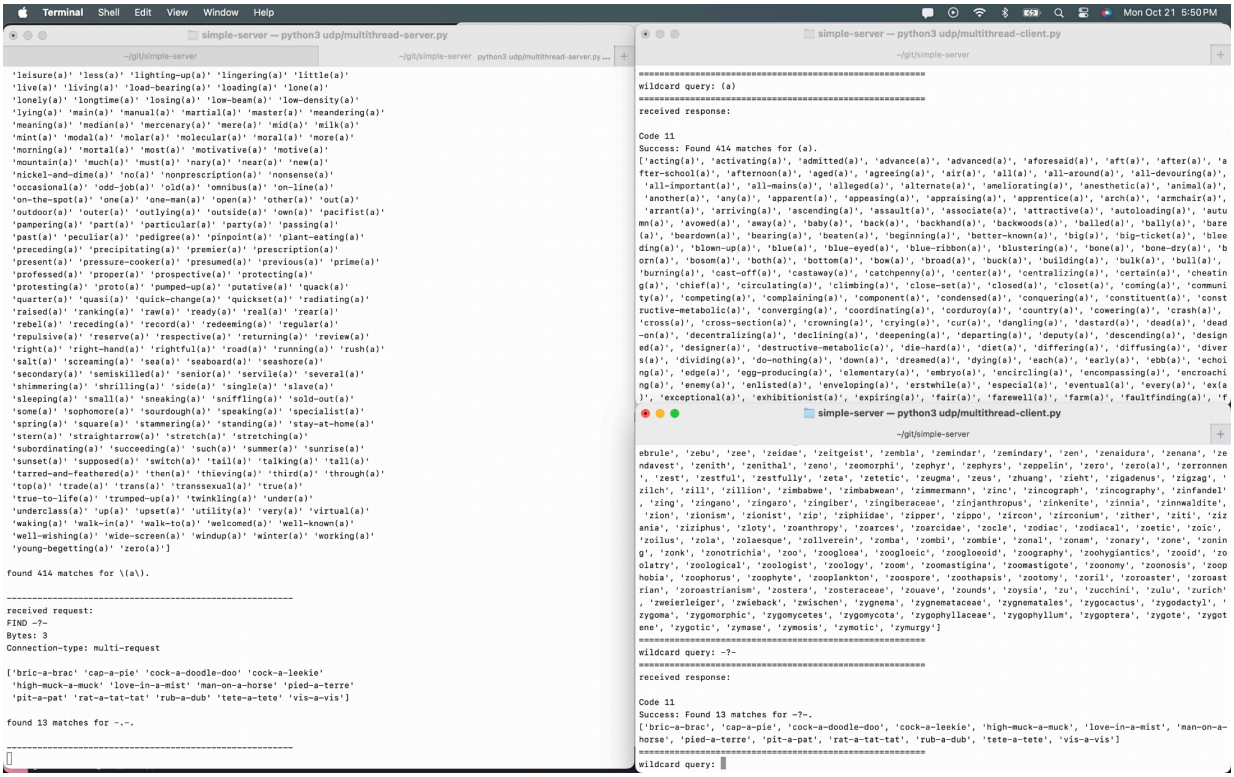
?????????? | ?

Test server response for 10 wildcard characters and for 1 wildcard character.



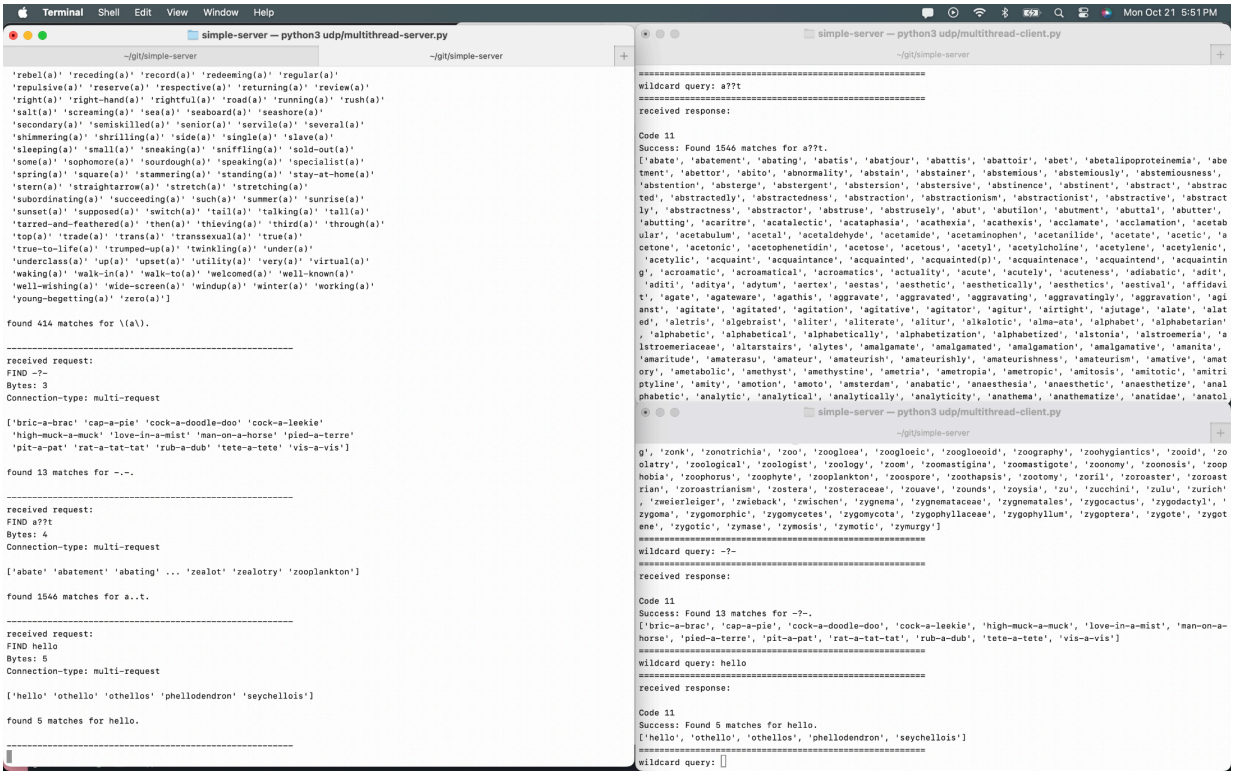
(a) | -ʔ-

Test server response for a query including special characters (parentheses and dashes).



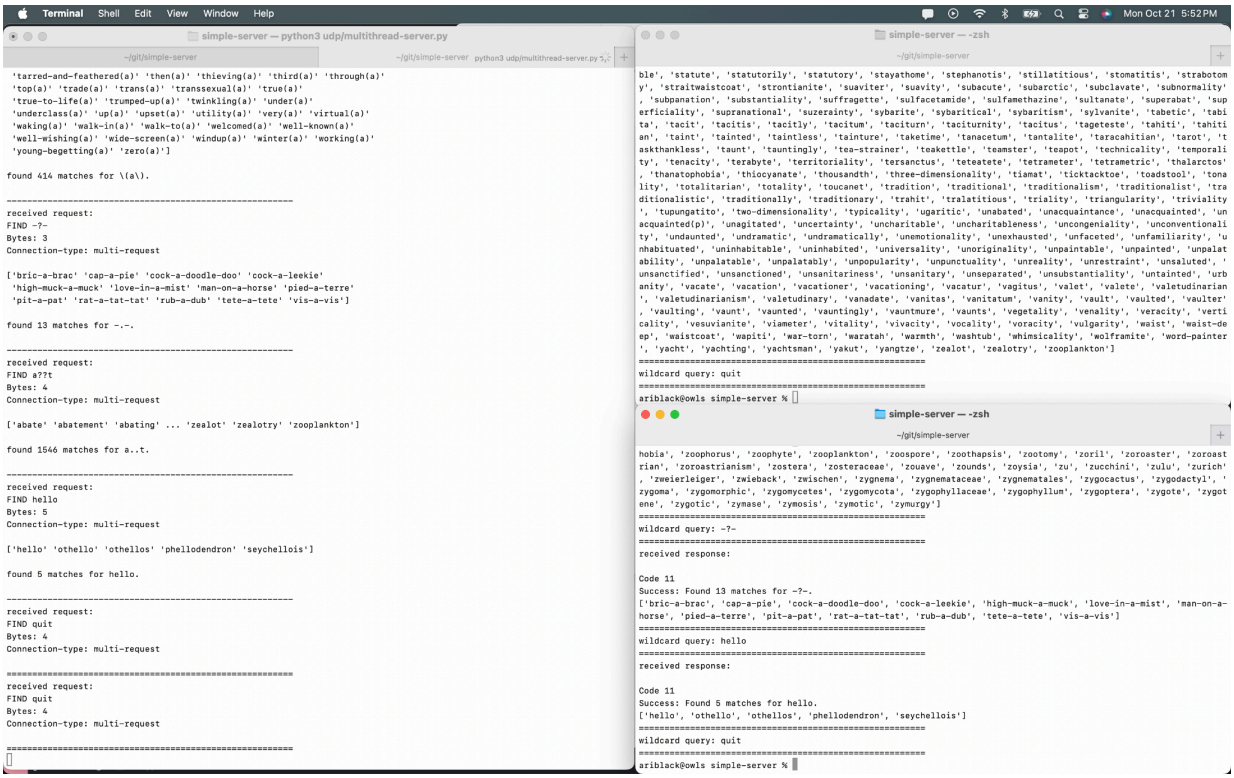
a??t | hello

Test server response for general case queries (1) including multiple wildcard and non-wildcard characters, and (2) including no wildcards.



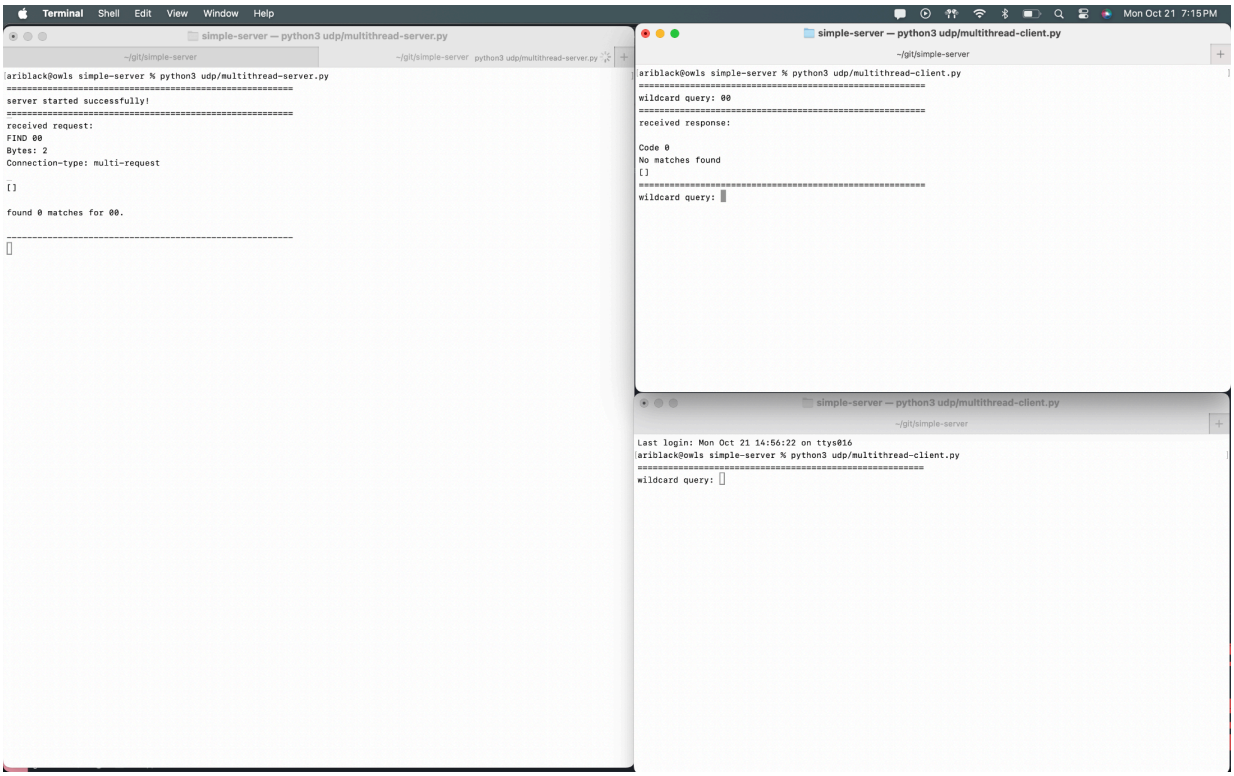
quit

Test server response for client quit, or termination of communications.



00

Test server response for a query that has no matches.



Known Issues: The major known issue with the UDP client/server programs is similar to the final issue listed in the TCP design document – sending non-ASCII characters may cause an error in the server (but *?*, -, and *()* are okay).

Resources

The code used in the *dispatcher()* method in *udp/multithread-server.py* is heavily influenced by some sample code accessible at the following link: https://pythontic.com/socketserver/threadingudpserver/introduction. Specifically, the code structure for iterating through each thread, appending the thread to a threads list, starting the threads process, then joining all threads immediately afterward was adopted from the listed source.

The remainder of the code (and all documentation) throughout the project was developed independently.