Networks Assignment

Maxi Clayton Clowes

4/12/2014

This assignment was to create a three

Using the code

Note: Code is in Python version 3.4.2

The code consists of two Python files, *Client.py* and *Server.py*. Each *client* must be run from a directory containing a server database (*serverdb.p)*, which keeps a list of all servers that the client will attempt to query. Each *server* must be run from a directory containing a movie database (*moviedb.p)* and database of valid ports *(portdb.p)*, which stores a list of permitted welcome ports that a server can use. These databases are implemented in the *pickle* Python module, which I opted for rather than simply including an array in the server file as it allows testing with differing database files, as well as saving/maintaining the database.

Upon starting, a client will print a list of commands with descriptions of their functionality. The client functionality is divided into two sections:

- Movie requests:
    - Get movie information – *"get"*
    - Edit movie information – *"edit"*
    - Add movie to database – *"add"*
    - Get list of all movies – *"all"*
- Server maintenance:
    - Add new server to list – *"addserver"*
    - Delete server from list – *"deleteserver"*
    - Print list of all servers client currently accesses – *"serverlist"*

The system

This system is based on the TCP protocol, which I selected for two main reasons: firstly, because the nature of the client requests is single data exchanges (the client sends a request with associated data as a single string and the client then sends a response); secondly, for the reliability that TCP offers – UDP would require more exception handling, should a response not be received from the server for instance.

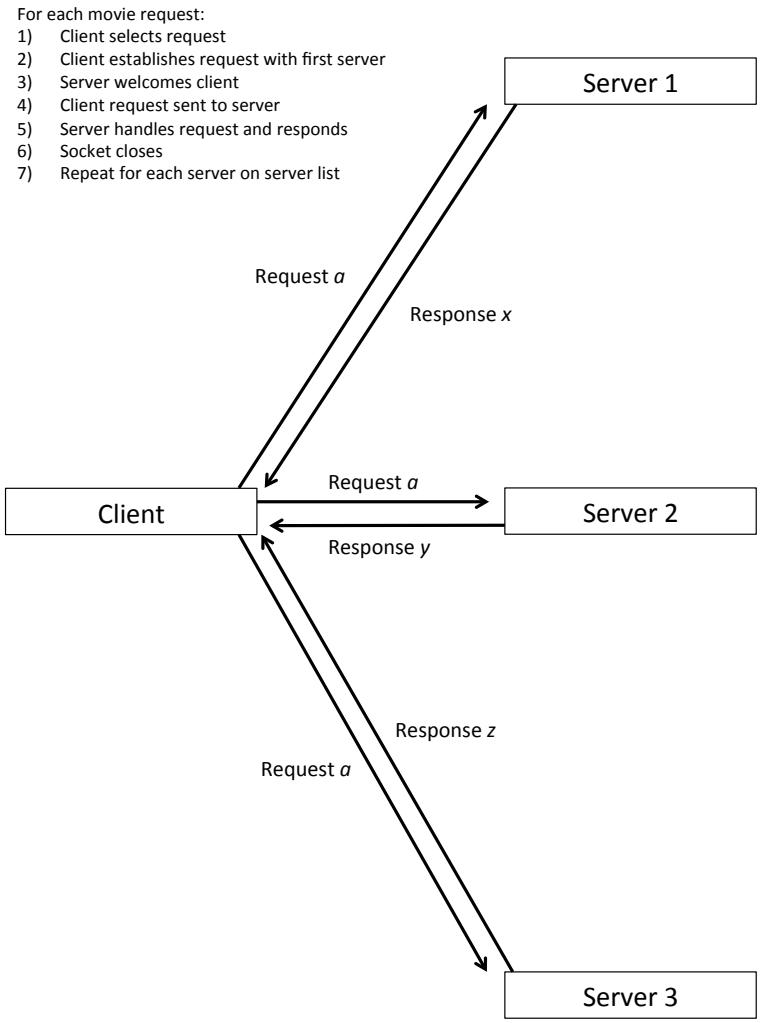
Upon running *TCPServer.py*, the server will iterate through the ports listed in it's *portdb.p* file, and setup a welcome port at the first available port. It will listen at this port for a client.

Upon running *TCPClient.py*, a list of the valid commands is printed (as seen above). The included list of servers are all at *localhost* as testing was based on a single machine; however, new servers at different IP addresses can be added with *addserver*.

When the client selects a movie request such as *get*, which takes a movie name and retrieves the associated information from servers, the client iterates through all server addresses in it's *serverdb.p* and attempts to connect to a socket at that address. If a socket is found and connection is established, the client sends a query and awaits a response from the server.

When a client successfully connects to a server, the server moves the client to a new socket and handles the exchange in a new thread (meaning that multiple clients and multiple requests can be handled by a single server with a single welcome socket), and goes back to listening at the welcome port. The server awaits a query from the client, handles the query, and sends a response to the client. After the exchange is complete the socket is closed and the thread terminates.

The client will send the same request to all servers and print all responses because the fact servers maintain independent databases means that a movie may exist with different information in each one, and with inconsistent information. The edit and add movie methods update the information in all servers that contain a movie of that name, making the databases consistent, and add a movie and it's information to any database without an existing entry by that name.



Potential errors and query failures are reduced at the data entry stage by keeping string format consistent across all entries; for example, all movie names are capitalized when adding/ searching.