

Kwaku Owusu - 109181846

Devon Maguire - 109284550

CSE 310 DNS Project - Dec 4th 2015

Overview:

Our project encompasses the entire assignment with support for two name servers, a manager to handle both of those servers, and the client. The manager stores the information of the two name servers, and each server stores the database for their respective type. The client can access the manager to get information about the servers, and then access the servers to get information about the domains within that type.

User Documentation:

To start the system first navigate to the directory where the program files are, then run 'python manager.py' which will print the port number the manager is using to listen for clients. Run 'python client.py HOSTNAME PORT' where HOSTNAME is the address of where the manager is running from (IP) and PORT is the port number the manager is listening on. Once the client program has connected you may use the command 'browse' to look at which type name servers are available. Then use the command 'type TYPE', where TYPE is any of the types listed in the browse command, to get the IP address and port number for the server corresponding to the specified type. After getting the server information, the client can disconnect from the manager using 'exit' and reconnect to the server by running 'client.py HOSTNAME PORT' again with HOSTNAME being the IP address of the server and PORT the port the server is listening on. Once connected to the server, the client can use the following commands:

'put NAME VALUE' where NAME is the name of the entry to be created (such as google.com) and VALUE is the address of where NAME is located (such as 127.0.0.1). If the entry is new the client will print "PIN: NAME entry inserted" if the entry already exists it will update the entry and print "PUP: NAME entry updated".

'get NAME' returns the VALUE field, or the address, corresponding with the entry that has name NAME. The client will print "ENT: The address of NAME is VALUE" upon success. Possible errors are "ERR: NAME not found", which means the the entry with name NAME could not be found in the database, and "ERR: The database is currently empty" which means the database does not have any entries which is why NAME could not be found.

'del NAME' deletes the entry from the database that has the name NAME. The client will print "EDE: NAME entry deleted" upon success. Possible errors are the same as those for get,

“ERR: NAME not found”, NAME is not in the database, and “ERR: The database is currently empty” which means the database is empty.

‘browse’ lists all the entries currently in the manager or database. If the client is connected to the manager, upon success the client will print the heading “--Types--” and a list of all types supported by the various servers. For the servers the client will print “----Name----” and a list of all the names currently in the database of that server. If there are no entries in that server the client will print “ERR: The database is currently empty.”

At anytime the user can also use the command ‘help’ to print a help menu for the client, manager, and servers, and the command ‘exit’ to exit the client. Please note that when exiting the client it will print to screen “Closing connection to server and closing program...”. Alternatively when a user is done with the manager or a server they can also type in the command ‘done’ which will terminate the connection as well.

As the client sends requests to the manager or the servers the manager program will print out the messages received by all processes, as well as when a new user connects. So if the client sends a browse request to the manager the manager will print “Client > Manager: 444 BRS” where 444 BRS is the protocol syntax for requesting the browse list from the manager. Similarly if the client connects to a server and then sends a browse request the manager will print “Connected to user at IP address: ADDRESS” and then “Client > Server: 444 BRS”. Messages received by the client are translated and printed out on the client screen.

Only 4 nameservers are supported with numerous clients able to access them. The manager can support numerous clients as well. One important thing to note is that these programs can only be run on unix based machines, meaning they cannot run on windows.

System Documentation:

The CMS protocol is our protocol between the client, manager, and servers. Each interactions starts with the client sending a “111 CON” message, or a connection request message. The acknowledgement for this request is “200 COS” or the connection successful response. All requests that are made by the client have repeating digits, so the different commands are handled as follows:

‘put’ is “222 PUT NAME VALUE” where NAME and VALUE are the arguments of the put command. The server then takes this information and either creates a new entry for which it sends the response “606 PIN NAME” to let the client know it inserted the entry NAME. Or it sends the response “607 PUP NAME” to let the client know it updated the entry NAME.

'get' is "333 GET NAME" where NAME is the name of the entry that the client wants to get the address of from the server. The server searches for the NAME and returns "610 ENT NAME VALUE" upon a successful find.

'browse' is "444 BRS" and takes no arguments, in this case server responds with "611 AEN HEADING\nNAME\nNAME\n..." where AEN stands for All Entries and the HEADING is the heading for the list of names all separated by newline characters. This is a convenient acknowledgement as the printing format is already set and can be sent directly to stdout by the client.

'del' is "555 DEL NAME" with NAME being the name of the entry to be deleted from the server. The server searched for NAME and upon a successful find deletes the entry. It then returns the acknowledgement "608 EDE NAME DELETED" to indicated that the entry with NAME has been deleted.

'type' is "777 TYP TYPE" where TYPE is the type of server the client wants information for from the manager. If the manager finds a server that handles type TYPE then it returns "613 TYF HOSTNAME PORT" where HOSTNAME is the address or hostname of the server and PORT is the port number it is listening on. The client can then print this out in a user friendly way on its side of the connection.

Finally 'done' and 'exit' both send "888 DNE" to indicated that they are done with the server or manager and are planning to exit so they can either stop, or reconnect to another server. The client waits for the "889 DNE" acknowledgement from the server and then both the server and the client close the sockets they were communicating on. The client then exits and the server remains active listening for other clients.

Throughout these connections there are a number of errors that can go wrong. All error messages are prefaced with the error code and the "ERR" message. Connection errors are all in the thousands, database errors are in the six-hundreds with the exception of it's not found errors. All not found errors are in the five-hundreds. For more specific error information please refer to the Protocol Codes table at the end of this document.

Windows does not support the fork() system call which is used in the manager.py program in order to run the servers, this program will only work on unix based machines. To run the manager you must navigate in the command line to the directory where the file is located. This directory should also hold the server.py, record.py, and manager.in files. Type in the command "python manager.py" and press enter to start the manager. This will then startup the servers as concurrent processes. Then as the clients connect to the manager and the server new

threads are created to handle them individually. To keep the threads from accessing the same data at the same time we use the thread library's lock function. This makes it so any code between the lines `lock.acquire()` and `lock.release()` can only be used by one thread at a time, keeping clients from creating race conditions by say browsing the server at the same time that someone else is deleting something.

The database files created by each server use python's pickle module. The `pickle.dump()` method allows objects to be quickly inserted into a file, by turning each object into a machine readable byte stream. The `pickle.load()` method then reads that byte stream and re-creates the objects. This greatly improves the efficiency of handling database entries.

Most of the error conditions in the program are created by bad user input, for example if the `client.py` program is run with a bad hostname or port number, then the program will tell the user that the connection cannot be made due to bad input. Similarly if a bad command is entered into the client that the server or the manager does not recognize it will print an input error letting the user know that the command is not valid and to try again. There are also exceptions that are thrown inside the programs that the user never sees such as IOE errors for a closed socket connection. If someone pressed control-c while a client is running the manager or server won't print out a bunch of information about the interrupt, instead it will handle it nicely.

Finally in order to compile and generate the executables for the programs you run the following programs in the command line of a machine that has Python 2.7 installed on it. For the manager run "`python manager.py`" which will create the executable and run it automatically. For the client run "`python client.py HOSTNAME PORT`" where HOSTNAME is the IP address or hostname of the manager and PORT is the portnumber the manager is listening on (9550).

Testing Documentation:

Scenario: Running and Connecting to the Manager

1. Upload your copy of `manager.py`, `manager.in`, and `server.py` to the same directory on any computer.
2. In the `manager.in` file, if no entries are inserted enter 4 DNS types, such as A, APL, NS, and MX, each followed by a newline.
3. On the same computer obtain your IP address and change the IP address in the programs to one found on the computer. For example you could use `allv24.all.cs.stonybrook.edu`.

4. Open up a command prompt and navigate to the directory you placed the python files in, and run the manager program by entering “python manager.py”. The program will now give you its port number 9550.
5. Place the client program anywhere on your computer and run it by entering python client.py followed by the IP address of the computer running the manager, and the port number
For Example: “python client.py allv24.all.cs.stonybrook.edu 9550”. The manager will print out a message with the IP address of the client as well as the connection message the client sent.
6. If you enter the command “browse” to show a list of all servers, you’ll see it will print out the list of all the servers you put in the manager.in file on the client side. On the manager side you’ll see the 444 BRS message that the client sent to the manager.
7. Enter “type APL” to get the port number of the server holding all DNS type APL records as shown in the screenshot below. Again on the manager you’ll see it received the request by the printed message “Client > Manager: 444 BRS”
8. Enter “exit” to disconnect from the manager and it will receive the “888 DNE” message to disconnect the client.

Client Screenshot:

```
Devons-MacBook-Pro:part2 Devon$ python client.py allv24.all.cs.stonybrook.edu 9550
COS: Succesful Connection
>browse
Types
-----
A
APL
NS
MX
>type l
ERR: l not found
>type APL
Please connect to allv24.all.cs.stonybrook.edu using port number 40811 to access the requested type.
>exit
Closing connection to server and closing program...
Devons-MacBook-Pro:part2 Devon$
```

Manager Screenshot:

```
allv24:~/project/part2> python manager.py
Manager is ready for connections on port 9550
Connected to a user at IP address: 130.245.207.103
Client > Manager: 111 CON
Client > Manager: 444 BRS
Client > Manager: 777 TYP l
Client > Manager: 777 TYP apl
Client > Manager: 888 DNE
```

Why this was chosen: This scenario will show that the manager is capable of handling connections, and creating/running multiple servers.

Scenario: Connecting to the Server and Entering Commands

1. After the manager gives a port number run the client again by entering python client.py followed by the host name and port number “python client.py allv24.all.cs.stonybrook.edu 10992”

```
Devons-MacBook-Pro:part2 Devon$ python client.py allv24.all.cs.stonybrook.edu 40811
COS: Succesful Connection
>
```

2. Enter the command “browse”, this shall give error, since the database has not been created.

```
>browse
ERR: The database is currently empty
>
```

3. Enter the command “get google.com”.

4. This will give you an error due to google.com not existing in the database.

```
>get google.com
ERR: google.com not found
>
```

5. Enter the command “put google.com 216.239.32.0”.

6. This will insert the record into the database.

```
>put google.com 216.239.32.0
PIN: google.com entry inserted
>
```

7. Enter the previous commands “browse” and “get google.com”, which will now show you google’s entry in the database and return the IP address for google.com.

```
>browse
----Name----
google.com
>get google.com
ENT: The address of google.com is 216.239.32.0
```

8. Enter the command “put google.com 8.8.8.8” and enter “browse”.

9. Google’s entry in the database has now been updated.

```
>put google.com 8.8.8.8
PUP: google.com entry updated
>browse
----Name----
google.com
```

10. Enter the command “del google.com” this will delete googles entry from the database.

11. Enter the command “browse” again, no entries will be shown.

```
>del google.com
EDE: google.com entry deleted
>browse
ERR: The database is currently empty
```

12. Enter the command “put google.com 8.8.8.8”.

13. Now exit the program by entering “done”.

14. Re-connect to the server the same way as you did before.

15. Now type “browse”.

16. Google’s entry has remained, proving the server’s persistence.

```
>put google.com 8.8.8.8
PIN: google.com entry inserted
>done
Devons-MacBook-Pro:part2 Devon$ python client.py allv24.all.cs.stonybrook.edu 53689
COS: Succesful Connection
>browse
----Name----
google.com
```

17. Enter a random string “I love CSE 310”.

18. The program will not recognize this as a command and no action will be taken.

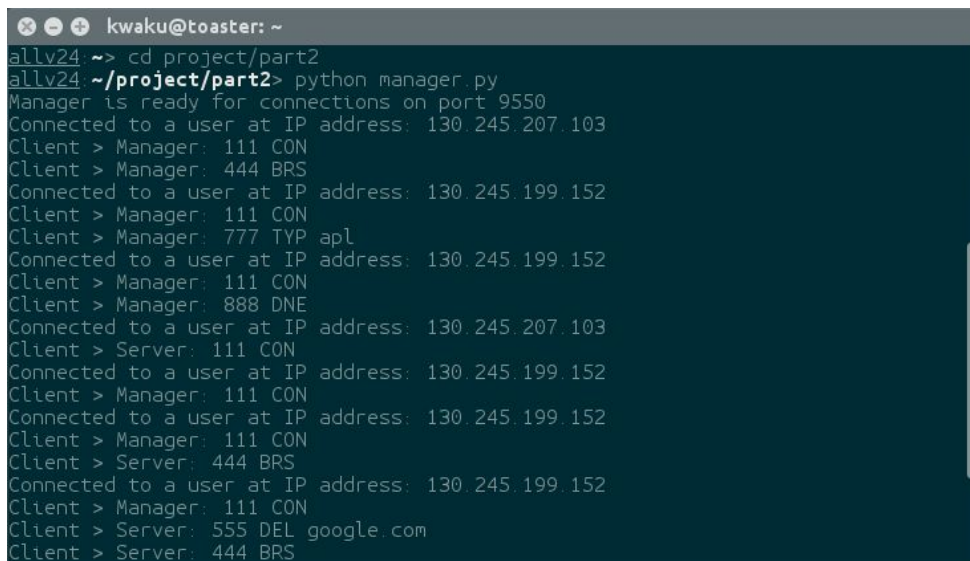
```
>I love CSE 310
Input Error - Please input a proper command
>
```

19. The same instructions can be repeated with any server, as long as you have the proper port.

Why this was chosen: This scenario shows that all user commands are functioning across the server and client. It also makes it evident that the server is able to write and read from its respective database file.

Scenario: Potential Race Condition Handling

1. Open up multiple terminal windows and connect to a name server using the same method listed above.
2. The server will be able to handle these connections without disruption, and will list information about each connected client.



```
kwaku@toaster: ~
allv24 ~-> cd project/part2
allv24 ~/project/part2> python manager.py
Manager is ready for connections on port 9550
Connected to a user at IP address: 130.245.207.103
Client > Manager: 111 CON
Client > Manager: 444 BRS
Connected to a user at IP address: 130.245.199.152
Client > Manager: 111 CON
Client > Manager: 777 TYP apl
Connected to a user at IP address: 130.245.199.152
Client > Manager: 111 CON
Client > Manager: 888 DNE
Connected to a user at IP address: 130.245.207.103
Client > Server: 111 CON
Connected to a user at IP address: 130.245.199.152
Client > Manager: 111 CON
Connected to a user at IP address: 130.245.199.152
Client > Manager: 111 CON
Client > Server: 444 BRS
Connected to a user at IP address: 130.245.199.152
Client > Manager: 111 CON
Client > Server: 555 DEL google.com
Client > Server: 444 BRS
```

3. All though this may be difficult, have someone else connect to the server and enter the command put google.com 8.8.8.8 in one terminal, and put google.com 216.239.32.0 in another terminal at the same time.
4. Due to the implementation of locks, and threads. No errors will be given to the clients. The entry will either be updated or inserted, depending on which record the server wants to insert first.
5. As with the previous example, entering del google.com and put google.com 8.8.8.8 at the same time will either result in an entry not found followed by entry inserted message, or an entry inserted followed by an entry deleted message.

Why this was chosen: In this case, multiple users will connect to the manager or server at the same time. Due to the implementation of threading, hanging is not experienced by users who are waiting to connect. The integrity of the database file is also preserved due to the implementation of locks, since multiple users can not write to the same file at once.

Protocol Codes:

C > M/S	MESSAGE	ACTION	M/S > C	MESSAGE	ACTION
111 CON	n/a	Request a connection with the server or the manager	200 COS	n/a	Connected successfully with the server or manager
			1116 ERR	n/a	Connection failure due to a bad hostname or port
			1229 ERR	n/a	Connection failure due to a timeout event
222 PUT	NAME VALUE	Request that a new entry be put into the name server	606 PIN	NAME	The entry with NAME has been inserted into the database
			607 PUP	NAME	The entry with NAME has been updated in the database
333 GET	NAME	Request the VALUE of entry with name NAME in the database	610 ENT	NAME VALUE	Return the name and value of the specified entry
			505 ERR	NAME NOT FOUND	Returns the name of the entry and says it cannot be found
			609 ERR	DATABASE EMPTY	The database is empty which is why the name cannot be found
444 BRS	n/a	Request for a list of	611 AEN	NAME1\nNAME2	Returns a list of all

		all the names currently on the server		\n <OR> TYPE1\nTYPE2\n	the names currently on a server or all the types currently supported by servers through the manager
			609 ERR	DATABASE EMPTY	The database is empty which is why the list cannot be returned
555 DEL	NAME	Requests to delete an entry NAME off of the name server	608 EDE	NAME DELETED	The specified entry has been successfully deleted
			505 ERR	NAME NOT FOUND	The entry cannot be found and is not deleted
			609 ERR	DATABASE EMPTY	The database is empty and therefore nothing can be deleted
777 TYP	TYPE	Request for information on the specific server that handles type TYPE	613 TYF	HOSTNAME PORT	The manager returns the port and hostname that the server handling type TYPE is running on
			511 ERR	TYPE NOT FOUND	The manager cannot find a server currently handling type TYPE
888 DNE	n/a	Leaving the current nameserver to go back to the manager	889 DNE	n/a	The server closes the connection with the client