CSE310_DNS_Project

Authors

- Dan Harel (109074254)
- Noel Sicoco (109265689)

Storage

DNS records are stored in a text file. Records are stored as follows:

`records.txt`:

```
name1 type1 value1\r\n
name2 type2 value2\r\n
```

Protocol

Request Format

```
DNS/1.0 METHOD name type value
```

name, *type*, and *value* are optional depending on the method.

Valid Methods

Server Only

- `PUT <name> <type> <value>`
- `GET <name> <type>`
- `DELETE <name> <type>`
- `BROWSE`

Manager Only

- `TYPE <type>`

Requests must be under 1024 bytes in length.

Response Format

```
DNS/1.0 StatusCode StatusPhrase\r\n
name1 type1 value1\r\n
name2 type2 value2\r\n
...
\r\n
```

Status Code	Status Phrase	Description
200	OK	Request succeeded, requested record(s) later in this message.
201	Created	Request succeeded, supplied record successfully created.
400	Bad Request	Request message not understood by server.
404	Not Found	Requested record not found on this server.
503	Service Unavailable	Server is currently unavailable to handle the request.

User Documentation

Dependencies

The program uses Python 2.7.

Running the program

First navigate to the directory containing the program files. Start the manager by typing

```
$ python manager.py
```

once the manager has started running, in a separate command line instance, you must then open the client by typing

```
$ python client.py
```

From here you can begin interfacing with the DNS program.

Commands

- help
  - Prints out the help menu
- put *name value*
  - Inserts a record stored in the nameserver with the given name and value.
  - Only usable when connected to a name server.
- get *name*
  - Prints out the name, type, and value of the record in the nameserver with the given name
  - Only usable when connected to a name server.
- del *name*
  - Removes the record in the nameserver with the given name.
  - Only usable when connected to a name server.
- browse
  - Prints out all records in the nameserver.
  - Only usable when connected to a name server.
- done

    - Closes the current connection with the nameserver and allows you to start a new connection.
    - Entries that are currently in the nameserver will persist.
    - Only usable when connected to a name server.
- type *type*
    - Establishes a connection with the name server corresponding to the given type.
    - only usable when not connected to a name server.

Limitations

None known

System Documentation

Command line arguments

The client uses the Python module argparse to handle command line arguments. Command line argument defintions are placed at the beginning of the main() function. Further documentation can be found here.

Adding user commands to the client

The client uses the Python module Cmd to handle user input. User commands are defined by methods that begin with "do_[command]", with the command arguments passed in as the second parameter. Further documentation can be found here.

Concurent operation handling

Concurrency is used in order to allow multiple users to use the application at the same time. Concurrency is implemented in all locations using multithreading. It's used in the following locations:

- manager.py
    - Allows multiple instances of the manager to be running simultaneously.
- server.py
    - Allows multiple instances of each name server to be running simultaneously.

Concurrency is also handled in each server instance when multiple requests (each on their own thread) try to access the stored records.

- A thread lock is used to prevent multiple requests from accessing the file at once. When one request thread acquires the lock, other request threads that want to acquire the lock will block until the lock is released.

Major data structures

The list of active name servers is stored in manager.py using a dictionary. It maps the server type to a 2-tuple containing the hostname and the port number of the server.

Client architecture

The client keeps track of two sockets: the current socket that the client is communicating with and the manager socket. The "current" socket may be the manager socket at times. The "current" socket is stored as the *sock* variable in DNSClient. The manager sock is stored in *manager_sock* variable.

Manager architecture

The manager first reads `manager.in`, which contains a type on each line. For each non-empty line, a new server is started in a new thread, and its address and port are stored in a 2-tuple and mapped to the type in a dictionary.

The manager then starts its own server on port 4254 in a new thread. This server handles each connection on a new thread. When it receives a `DNS/1.0 TYPE` request, it returns the address and port of the DNS server mapped to the requested type.

Server architecture

Each server is started on an ephemeral port.

Each server runs on its own thread and handles requests by creating a new thread for each request. Each request is validated by checking the protocol (`DNS/1.0`), the request (`PUT`, `BROWSE`, etc.), and the other arguments. Then the request is processed, the file lock is acquired (or the thread waits until it can acquire it), the file is read/written to, the file lock is released, and lastly the server responds to the client with the appropriate status code and data.

Extra `.py` Files

ip_address.py

- Used by `manager.py` and `server.py`.
- Contains a function which returns the ip address of the machine.

ThreadedTCPServer.py

- Used by `manager.py` and `server.py`.
- Contains the class `ThreadedTCPServer` which is used to create a socket server that handles requests on separate threads.

Testing

Assumptions

Unless otherwise stated, the following assumptions are made for each test case: +There is a *manager.in* file with the following entries: `plaintext A NS CNAME` +The manager is running in a separate terminal window or background process, binded to hostname *hostname* and port *port*.

+Each name server is emptied at the end of each test case. This assures that each test case can be run independently, and that the success or failure of the previous test does not affect the next test.
+All running clients are closed at the end with the "exit" command.

Test PUT

Input

```
$ python client.py _hostname_ _port_
> type A
> put example.com 1.2.3.4
> put example2.com 5.5.5.5
> put example3.com 100.100.100.100
> put example.com 2.2.2.2
> browse
```

Expected output

```
Name Type Value
example2.com a 5.5.5.5
example3.com a 100.100.100.100
example.com a 2.2.2.2
```

Explanation

By inserting records then using "browse", we can see that they were correctly inserted. We can also see that we can use the PUT command to change the value of a record.

Test GET

Input

```
$ python client.py _hostname_ _port_
> type A
> put example.com 1.1.1.1
> put example2.com 2.2.2.2
> put example3.com 3.3.3.3
> get example2.com
```

Expected output

```
Name Type Value
example3.com a 3.3.3.3
```

Explaination

This shows that an entry's value can be retreived, regardless of how many entries have been placed

before or after the requested entry.

Test DEL

Input

```
$ python client.py _hostname_ _port_
> type A
> put example.com 1.1.1.1
> put example2.com 2.2.2.2
> put example3.com 3.3.3.3
> del example2.com
> browse
```

Expected output

```
Name Type Value
example.com a 1.1.1.1
example3.com a 3.3.3.3
```

Explaination

This shows that an entry's value can be deleted, regardless of how many entries have been placed before or after the requested entry.

Test manager concurrency

Input

```
$ python client.py _hostname_ _port_
Open new terminal window
$ python client.py _hostname_ _port_
> type A
Open the first terminal window
> type NS
```

Expected output

The second and third windows should both say `Connection established with nameserver`.

Explaination

By opening two connections at the same time, we see that both connections can still be used, and that the existance of one connection does not affect the others.

Test client concurrency

Input

```
$ python client.py _hostname_ _port_
Open new terminal window
$ python client.py _hostname_ _port_
> type A
Open the first terminal window
> type A
> put example.com 1.1.1.1
Open the second terminal window
> put example2.com 2.2.2.2
> browse
Open the first terminal
> browse
```

Expected output

Both clients should output the following:

```
Name Type Value
example.com a 1.1.1.1
example2.com a 2.2.2.2
```

Explaination

This shows that both clients can communicate with the same name server at the same time without affecting the other connection, and also receive the same output.