

Task 6.2C My DNS Server

Step 1

- ✓ The first step I took was to review well the work criteria to ensure that I had the best understanding of what was required of me. The required steps were to create DNS server and client only in python and without using any pre-made DNS packages. Therefore, the specific requirements are: 1 to support Hostname-to-IP-address translation, 2 host aliasing, 3 either A or CNAME DNS record types, and 4 it has to use UDP.

Step 2

- ✓ I first built the map for I then had to set it aside for the purpose of storing the DNS records. This dictionary contained the following information: the used hostname, the corresponding value and, finally, the type of DNS record – A or CNAME. An alias.com record would be 'alias.com': An MX record would be example.com : {'type': 'MX', 'value': 'mail.example.com'} while an A record is example.com : {'type': 'A', 'value': '93.184.216.34'}.

Step 3

When I wrote the DNS server code, I made sure it included the following features:

- ✓ **Socket Creation:** To achieve this I used second variable called data_socket which is a UDP socket and I used the bind method from socket module of Python to bind the socket to the given port and address. Conflicts are another recurring consideration and to minimize them, I initially choose my DNS port to be 53 and later on switched to 8053.
- ✓ **Handling queries:** In order to handle petitions I established a handle_query method that from an HList decomposes the query, decoded it in order to obtain the hostname and the type of record requested and proceed to verifying the records of the DNS and configured a way to reply. In case the CNAME was followed by an A record the server replaced the CNAME with the A record IP address.
- ✓ **Listening for Inquiries:** The waiter eagerly waited for new questions to pop up. It addressed each one using the handle_query method and returned the relevant result.

Step 4

I then developed the DNS client code that manipulates the server.

- ✓ **Sending Queries:** The client then passed the host name and record type entered by the user to the server through a UDP socket after setting the query.
- ✓ **Receiving Responses:** The query was provided by the server and after it, the client was shown the information by the user.
- ✓ **User Interaction:** The client continued to ask the user questions until the user gave BlackBox a command to cease its questions.

Step 5

When I ran the server code, I noticed that Port 53 was already in use and that I was getting an `OSError: [Errno 98] Address already in use`. To correct this:

- ✓ Port Modification: The port number 8053 for the server was modified and the `start_dns_server` function was also modified.
- ✓ Client Update: The client code was modified to meet the new server port number being 8053.

Step 6

I used the server and client to test the implementation:

- ✓ Launching the Server: To ensure that the server was running on port 8053 I ran the code `dns_server.py`.
- ✓ Starting the Client: I ran `dns_client.py` to the main from shell, entered the hostnames and record kinds and checked if the client is receiving correct responses from the server.

I'll now give you the client and server codes.

Server code

```
import socket

# Define a dictionary to hold DNS records
dns_records = {
    'example.com': {'type': 'A', 'value': '93.184.216.34'},
    'alias.com': {'type': 'CNAME', 'value': 'example.com'},
    'google.com': {'type': 'A', 'value': '8.8.8.8'},
    'alias2.com': {'type': 'CNAME', 'value': 'google.com'}
}

def handle_query(data):
    # Decode the query
    query = data.decode().strip()
    print(f"Received query: {query}")

    # Split the query to get hostname and record type
    parts = query.split()
    if len(parts) != 2:
        return "Invalid query format. Use: <hostname> <type>".encode()

    hostname, record_type = parts

    # Check if hostname is in DNS records
    if hostname in dns_records:
        record = dns_records[hostname]
        if record['type'] == record_type:
            response = f"{hostname} -> {record['value']}"
            # If it's a CNAME, resolve it to its final A record
            if record_type == 'CNAME' and record['value'] in dns_records:
                cname_record = dns_records[record['value']]
                if cname_record['type'] == 'A':
                    response += f" -> {cname_record['value']}"
            else:
                response = f"No {record_type} record found for {hostname}"
        else:
            response = f"{hostname} not found"

    return response.encode()

def start_dns_server(host='127.0.0.1', port=8053):
    # Create a UDP socket
```

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((host, port))
print("DNS server is running on port 8053...")
```

```
while True:
    # Receive data from client
    data, client_address = server_socket.recvfrom(512)
    response = handle_query(data)
    # Send response back to client
    server_socket.sendto(response, client_address)
```

```
if __name__ == "__main__":  
    start_dns_server()
```

[illegible]

Client Code

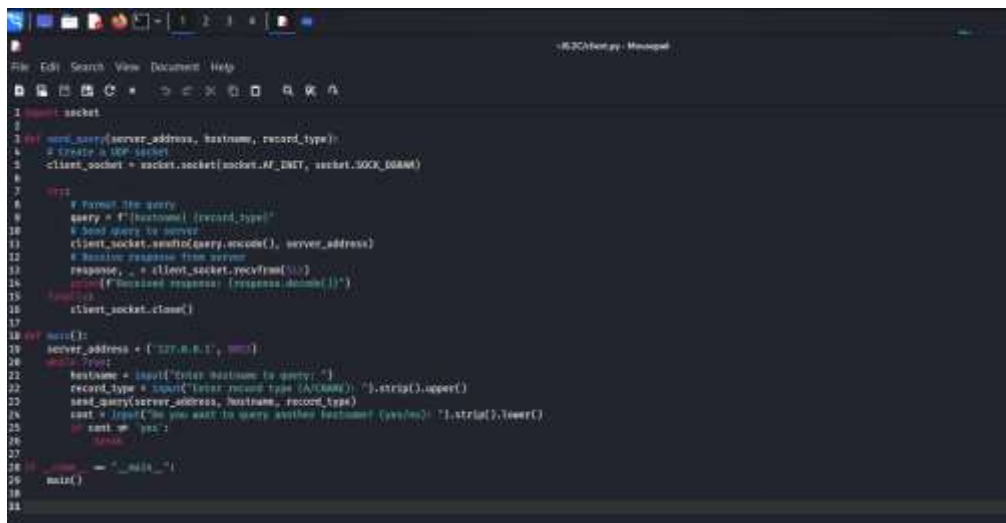
```
import socket

def send_query(server_address, hostname, record_type):
    # Create a UDP socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    try:
        # Format the query
        query = f"{hostname} {record_type}"
        # Send query to server
        client_socket.sendto(query.encode(), server_address)
        # Receive response from server
        response, _ = client_socket.recvfrom(512)
        print(f"Received response: {response.decode()}")
    finally:
        client_socket.close()

def main():
    server_address = ('127.0.0.1', 8053)
    while True:
        hostname = input("Enter hostname to query: ")
        record_type = input("Enter record type (A/CNAME): ").strip().upper()
        send_query(server_address, hostname, record_type)
        cont = input("Do you want to query another hostname? (yes/no): ").strip().lower()
        if cont != 'yes':
            break

if __name__ == "__main__":
    main()
```



```
1 import socket
2
3 def send_query(server_address, hostname, record_type):
4     # Create a UDP socket
5     client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6
7     try:
8         # Format the query
9         query = f"{hostname} {record_type}"
10        # Send query to server
11        client_socket.sendto(query.encode(), server_address)
12        # Receive response from server
13        response, _ = client_socket.recvfrom(512)
14        print(f"Received response: {response.decode()}")
15    finally:
16        client_socket.close()
17
18 def main():
19     server_address = ('127.0.0.1', 8053)
20     while True:
21         hostname = input("Enter hostname to query: ")
22         record_type = input("Enter record type (A/CNAME): ").strip().upper()
23         send_query(server_address, hostname, record_type)
24         cont = input("Do you want to query another hostname? (yes/no): ").strip().lower()
25         if cont != 'yes':
26             break
27
28 if __name__ == "__main__":
29     main()
30
31
```

I will now present the results of both codes.

Sever Output

```
kali@kali: ~/6.2C
File Actions Edit View Help
(kali@kali)-[~/6.2C]
$ python3 server.py
DNS server is running on port 8053 ...
Received query: example.com CNAME
Received query: kenisha.com A
Received query: alias.com CNAME
█
```

Client Output

```
kali@kali: ~/6.2C
File Actions Edit View Help
(kali@kali)-[~/6.2C]
$ python3 client.py
Enter hostname to query: example.com
Enter record type (A/CNAME): CNAME
Received response: No CNAME record found for example.com
Do you want to query another hostname? (yes/no): yes
Enter hostname to query: kenisha.com
Enter record type (A/CNAME): A
Received response: kenisha.com not found
Do you want to query another hostname? (yes/no): yes
Enter hostname to query: alias.com
Enter record type (A/CNAME): CNAME
Received response: alias.com → example.com → 93.184.216.34
Do you want to query another hostname? (yes/no): no
(kali@kali)-[~/6.2C]
$ █
```

Summary and reflection on this Task

Summary

- In this assignment, I developed a DNS server and client program on the Python programming environment. This was the goal of constructing a hypothetical DNS server capable of handling resource records of CNAME (canonical name) and A. The server required to answer queries over UDP, the client had to be able to ask questions to the server and receive answers in return.
- The DNS server doubles for questions coming its way by parsing the hostname and the record type, searching for the records and providing the appropriate response. Instead, the DNS server maintains DNS records in a dictionary. Both the hostname and record type were provided by the user and input into the client program and the request was sent to the server and the result was displayed.
- An error occurred: [Errno 98] When I first began getting problems with the default DNS port which is 53, the message showing was 'address already in use'. As a solution, I changed the server and client to use port 8053. By so doing, client-server interaction was enhanced and servers run effectively without conflicts.

Reflection

- On balance, it was a pleasant practice in terms of accomplishing educational objectives of the work. First of all I extended my knowledge with DNS protocols especially with the usage of UDP for DNS queries and responses. Since I had to implement the server part and the client part using my own algorithms not relying on any other DNS libraries it was quite difficult for me to get a complete grasp of how DNS operations work in details.
- After the port conflict issue was discovered and resolved, some of the activities among them were trouble shooting and problem solving. It emphasized the fact that more can be learned about the network settings and some leeway should always be provided since problems may occur.
- In my case, this challenge was useful for developing my overall Python programming, particularly in networks. It also extended my skills of critical thinking when designing and launching the network services that correspond to the defined protocols. It has been a quite useful project that has helped me build a clear understanding of DNS, which will be beneficial for more complex networking topics in future work.