Task 6.2C

Step 1

I started by carefully going over the task criteria to make sure I knew what was required of me. The assignment was to use Python to create a DNS server and client without the use of pre-existing DNS packages. The server has to enable hostname-to-IP-address translation, host aliasing, two types of DNS records (A and CNAME), and UDP communication.

step 2

To store the DNS records, I first defined a dictionary. The hostname, the associated value, and the type of DNS record (A or CNAME) were all saved in this dictionary. For example, 'alias.com': {'type': 'CNAME', 'value': 'example.com'} for a CNAME record and 'example.com': {'type': 'A', 'value': '93.184.216.34'} for an A record.

step 3

I included the following essential features in the DNS server code when I created it:

- Socket Creation: I used the socket module in Python to build a UDP socket and bound it to a port and address. I used 53, the default DNS port, at first, but I later changed it to 8053 to minimize conflicts.
- Managing inquiries: To decode incoming inquiries, extract the hostname and record type, check DNS records, and create a response, I implemented a method called handle_query. The server resolved a CNAME record to its final A record if one was discovered.
- Listening for Queries: The server was always on the lookout for new inquiries. It used the handle query method to handle each one and returned the relevant answer.

step 4

The DNS client code to communicate with the server was then written by me

- Sending Queries: Using a UDP socket, the client prepared the query and delivered it to the server after receiving input from the user for the hostname and record type.
- Getting Responses: The client sent the query, the server responded, and the client showed the user the information.
- User Interaction: Until the user made the decision to stop, the client kept asking the user questions.

step 5

Port 53 was already in use when I executed the server code and saw an OSError: [Errno 98] Address already in use. To fix this:

- Modifying the Port: I modified the start_dns_server function to use port 8053 on the server.
- Client Update: In order to conform to the new server port (8053), I modified the client code.

step 6

I tested the implementation by running the server and client:

- Starting the Server: I ran dns_server.py and confirmed the server was running on port 8053.
- Running the Client: I ran dns_client.py, entered hostnames and record types, and verified that the client received correct responses from the server.

Now I'll provide the server and client code

Server Program 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"
```

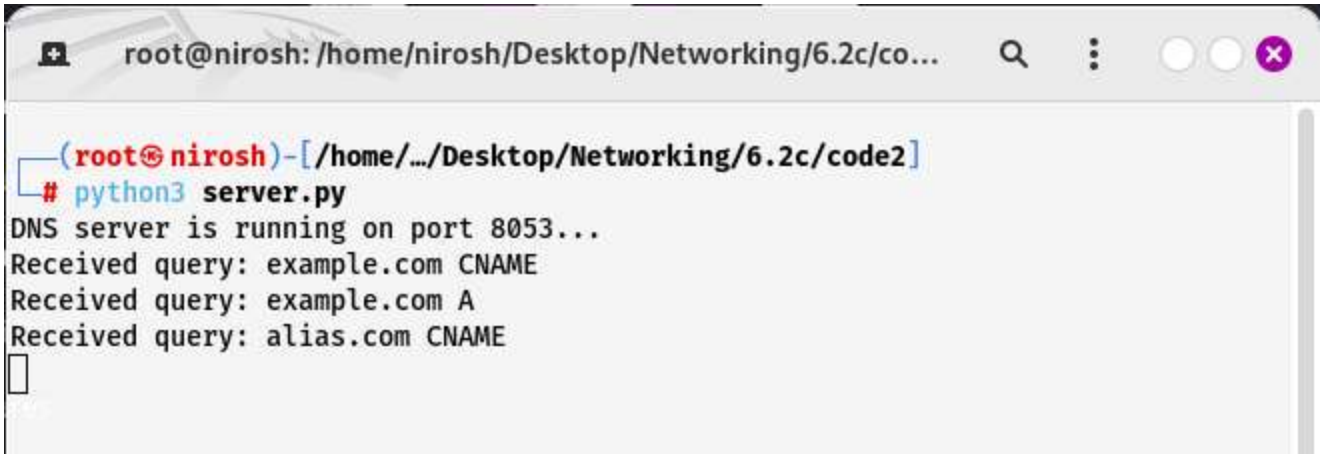
```
    return response.encode()

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()
```

Client Program code

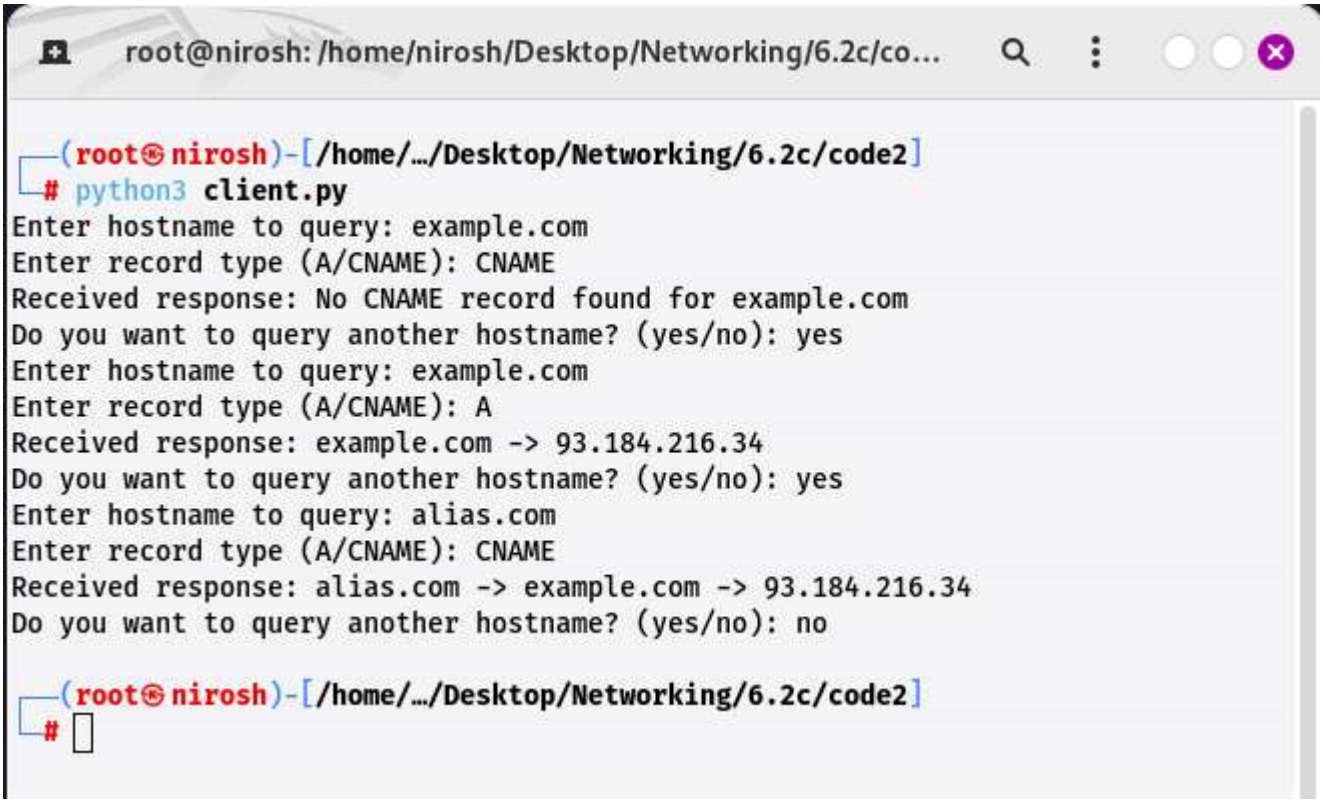
```
import socket

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()
```

Now I'll provide the outputs of both the code

Server output



Client output



Summary and Reflection on this task

Summary

I used Python to create a DNS server and client for this assignment. The goal was to build an imaginary DNS server that could handle both CNAME (canonical name) and A (address) resource records. The client was made to be able to send questions to the server and get answers, and the server had to handle inquiries over UDP.

The DNS server was designed to handle incoming inquiries by extracting the hostname and record type, looking up the records, and returning the relevant response. DNS records are stored in a dictionary on the DNS server. The hostname and record type were provided by the user into the client application, which then sent the query to the server and displayed the response.

An OSError: [Errno 98] Address already in use appeared when I first encountered a problem with the default DNS port (53), which was already in use. I changed the server and client to use port 8053 in order to fix issue. This modification guaranteed conflict-free server operation and efficient client-server communication.

Reflection

This assignment was a worthwhile educational experience in a number of ways. I learned more about DNS protocols in the first place, especially how UDP is used for DNS queries and responses. It was difficult for me to fully comprehend the fundamental mechanics of DNS operations because I had to implement the server and client from scratch without using any pre-existing DNS libraries.

The port conflict issue was encountered and resolved, providing a useful lesson in troubleshooting and problem-solving techniques. It emphasized how crucial it is to comprehend network settings and to be flexible in the event that unforeseen problems occur.

Overall, this challenge helped me become better in Python programming, particularly with networks. It also improved my capacity for critical thought when it comes to designing and implementing network services that fit with particular protocol specifications. My basic understanding of DNS has been reinforced by this practical effort, and it will provide a good platform for future work with more complex networking topics.

Some extra resources I referred to

```
blog.dnsimple.com.      CNAME   aetrion.github.io.
aetrion.github.io.      CNAME   github.map.fastly.net.
github.map.fastly.net.  A       185.31.17.133
```


1. *Differences Between A and CNAME records - DNSimple Help*. (n.d.). https://support.dnsimple.com/articles/differences-a-cname-records/
2. Brumby, P., & Brumby, P. (2022, October 26). *DNS records explained: A, AAAA, NS, MX and CNAME*. https://www.pbrumby.com/2018/05/09/dns-records-explained/
3. Awati, R. (2022, February 25). *canonical name (CNAME)*. SearchWindowsServer. https://www.techtarget.com/searchwindowsserver/definition/canonical-name