Task 5.2C My DNS Server Sketch

Task

1. Server Initialization: Outline the steps for starting a DNS server, including initialization of necessary components.

    - **Server initialization**
      In this stage, the required parts are also assembled, and the DNS server is set up. DNS typically operates using UDP for the fast handling of its queries; therefore, the server uses the User Datagram Protocol (UDP). Here the server socket is bound to the normal DNS port which is 53.

```
# Function to initialize the DNS server
def initialize_server():
    # Create a UDP socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # Bind to the local address (IP) and port
    try:
        server_socket.bind(("0.0.0.0", 53))  # Bind to port 53 (DNS)
        print("DNS server started and listening on port 53")
    except:
        print("Failed to bind to port 53")
        sys.exit(1)
    return server_socket
```

    - **Explanation**
      UDP Socket Creation: To create a UDP socket for network connection the server uses socket.socket(socket.AF_INET, socket.SOCK_DGRAM).
      Port Binding: In the same way that server_socket.bind(("0.0.0.0", 53)) makes the server listen for DNS queries on port 53, server = socketserver.TCPServer(("", 53), DNSHandler) does the same.
      Error Handling: If the server does not bind to port 53 then the server quits with an error message.

2. Listening and Processing DNS Queries: Describe how the server listens for and processes incoming DNS queries. Develop logic for parsing queries to identify the hostname and query type (A or CNAME).

- **Listening and Processing DNS Queries**
  Once the server has been started up, it listens for DNS requests that are coming in. The flags used in determining a transaction ID, the hostname to which the response belongs and query type (A or CNAME) can all be gotten from analyzing the DNS message that is received.

```
# Function to listen for incoming DNS queries
def listen_for_queries(server_socket):
    while True:
        # Receive a DNS query (max buffer size of 512 bytes)
        query, client_address = server_socket.recvfrom(512)ww
        print(f"Received query from {client_address}")
        # Parse the DNS query to extract the transaction ID, flags, hostname, and query type
        transaction_id, flags, questions, query_name, query_type = parse_dns_query(query)
        # Handle the query based on the type (A or CNAME)
        if query_type == "A":
            handle_a_record(query_name, client_address, transaction_id, server_socket)
        elif query_type == "CNAME":
            handle_cname_record(query_name, client_address, transaction_id, server_socket)
        else:
            send_error_response(client_address, transaction_id, server_socket)
```

- **Explanation**
  Receiving enquiries: The recvfrom function (512) is used by the server to listening to the enquiry and to collect up to 512 bytes of information from the client.
  Parsing Queries: parse_dns_query(query) returns the hostname and query type (A or CNAME), and many other essential pieces of information.
  Conditional Handling: For CNAME records, the server supports handle cname record whereas for A records, it can handle a record only.

3. Handling A and CNAME Records: Design separates logical flows for dealing with A and CNAME record queries.

- **Handling A and CNAME Records**
  For such reasons, the server uses different routes of logic to address A and CNAME record searches. For CNAME records, it gets the alias while for A records it gets the address.

```
# Function to handle A record queries

def handle_a_record(query_name, client_address, transaction_id, server_socket):

    dns_records = {

        "example.com": {"A": "93.184.216.34"},

        "test.com": {"A": "192.0.2.1"}

    }

    if query_name in dns_records and "A" in dns_records[query_name]:

        ip_address = dns_records[query_name]["A"]
        response = generate_dns_response(transaction_id, query_name, "A", ip_address)
        server_socket.sendto(response, client_address)
        print(f"Sent A record response to {client_address}")

    else:

        send_error_response(client_address, transaction_id, server_socket)

# Function to handle CNAME record queries
def handle_cname_record(query_name, client_address, transaction_id, server_socket):

    dns_records = {

        "alias.com": {"CNAME": "example.com"},
        "anotheralias.com": {"CNAME": "test.com"}

    }

    if query_name in dns_records and "CNAME" in dns_records[query_name]:

        cname_value = dns_records[query_name]["CNAME"]
        response = generate_dns_response(transaction_id, query_name, "CNAME", cname_value)
        server_socket.sendto(response, client_address)
        print(f"Sent CNAME record response to {client_address}")

    else:

        send_error_response(client_address, transaction_id, server_socket)
```

- **Explanation**
  Handling A Records: If the questioned domain contains an A record, the server get the associated IP address and launches the DNS response.

  Managing CNAME Records: When it comes to CNAME queries, the server pull off the alias domain, (CNAME) and generate the proper DNS response.

  Error Handling: When the domain is not found with the record database or is not valid, It results the server to return an error response.

4. Generating DNS Responses: Develop the process for creating and sending appropriate DNS response messages

    - **Generating DNS Responses**
      Last but not least, depending on the type of the query – it could be query type A or CNAME, the server then comes up with the DNS answer. There is a header, the questions' part and the answer part in the reply.

Answer Header

```
# Function to generate a DNS response
def generate_dns_response(transaction_id, query_name, record_type, record_value):
    # Construct the response header
    flags = b'\x81\x80'  # Standard query response, no error
    question_count = b'\x00\x01'  # One question
    answer_count = b'\x00\x01'    # One answer
    ns_count = b'\x00\x00'        # No authority records
    ar_count = b'\x00\x00'        # No additional records

    # Construct the question section (same as the query)
    question_section = generate_query_section(query_name, record_type)

    # Construct the answer section (A or CNAME)
    answer_section = generate_answer_section(query_name, record_type, record_value)

    # Return the full DNS response
    return transaction_id + flags + question_count + answer_count + ns_count + ar_count +
question_section + answer_section
```

Section Answer

```
# Function to generate the answer section for A or CNAME
def generate_answer_section(query_name, record_type, record_value):
    pointer_to_query_name = b'\xC0\x0C'  # Use label compression

    if record_type == "A":
        answer_type = b'\x00\x01'  # A record
        answer_value = socket.inet_aton(record_value)  # Convert IP to binary
    elif record_type == "CNAME":
        answer_type = b'\x00\x05'  # CNAME record
        answer_value = convert_name_to_dns_format(record_value)

    answer_class = b'\x00\x01'  # Class IN (Internet)
    ttl = struct.pack('!I', 300)  # 5 minutes TTL
    data_length = struct.pack('!H', len(answer_value))  # Length of the answer

    return pointer_to_query_name + answer_type + answer_class + ttl + data_length +
answer_value
```

- **Explanation**
  Answer Header: The header (flags = b'\x81\80') has proved one question and one answer of the last successful query.

  Section of Answer: In answer part the server gives the IP address (A record) or an alias (CNAME) if it is an A or CNAME request.