JULY 27, 2023

COMP3331/9331 Computer Networks and Applications
Assignment for Term 2, 2023

DNS Client Server Model

Build the basic version of Resolver and client.

Resolver

Module

- sys (Take the arguments)
- socket (For Client and server model)
- select (For Client Multiplexing)

Basic Structure:

1. First we create the main function and take the port as an argument.
2. We will check if the port is given if not then we will print the syntax and close the program.
3. Using the try and except statement we check if the port is valid or not.
4. If the port is valid then we pass the port to the function run_dns_server for forwarder processing.
5. In the run_dns_server, first we will create a socket object using the AF_INET and SOCK_DGRAM.

```
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python server.py <port>")
        sys.exit(1)

    try:
        port = int(sys.argv[1])
        run_dns_server(port)
    except ValueError:
        print("Invalid port number. Please provide a valid integer
        sys.exit(1)
```

6. We will then bind the ip of the resolver machine and port with the object.
7. We will then print that DNS server is running on N port.
8. We use the try and except statement for the error handling.
9. We will run a while loop in which we will create the connection of resolver with theclient and wait for the message from the client.
10. Once, we receive the message we will process it and send it back to the client.
11. After the connection has been completed, we will close the socket.

```
22
23  def run_dns_server(port):
24      server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25      server_socket.bind(("127.0.0.1", port))
26
27      print(f"DNS Server is running on port {port}...")
28      try:
29          while True:
30              # Use select to handle multiple clients
31              readable, _, _ = select.select([server_socket], [], [], 0.1)
32
33              for sock in readable:
34                  data, client_address = sock.recvfrom(1024)
35                  response = handle_dns_request(data)
36                  server_socket.sendto(response, client_address)
37      except KeyboardInterrupt:
38          print("DNS Server stopped.")
39      finally:
40          server_socket.close()
41
```

12. In the handle_dns_request, we will check if the domain matches with any record inthe list or our database.
13. We will get the ip of the domain that matches and send it back to the client.
14. If no domain is matched then we will return not found to the client.

```
# Define DNS records (replace with your own domain and IP mappings)
DNS_RECORDS = {
    "example.com": "192.168.0.1",
    "google.com": "8.8.8.8",
}

def handle_dns_request(data):
    domain = data.decode().strip()

    if domain in DNS_RECORDS:
        ip_address = DNS_RECORDS[domain]
    else:
        ip_address = "Not found"

    return ip_address.encode()
```

Client

Module

- sys (Take the arguments)
- socket (For Client and server model)
- time (timeout)

Basic Structure:

1. First we take resolver ip, port, domain and timeout(optional) as an argument and if any argument is not given then syntax is printed on the screen. Timeout is optional.
2. We check if the port, ip for the resolver and domain all of them are valid.
3. We will then save the resolver ip in the server_ip variable, port in the server_port variable and domain in the domain variable.
4. We send all the argument to the query_dns_server for connection and domainmessage sending to the resolver.
5. After we get a message return from the query_dns_server, we will print it on thescreen.

```
35  if __name__ == "__main__":
36      if len(sys.argv) < 4:
37          print("Usage: python client.py <server_ip> <server_port> <domain> <timeout(optional)>")
38          sys.exit(1)
39
40      server_ip = sys.argv[1]
41      try:
42          server_port = int(sys.argv[2])
43      except ValueError:
44          print("Usage: python client.py <server_ip> <server_port> <domain> <timeout(optional)>")
45          sys.exit(1)
46      domain_name = sys.argv[3]
47      ip_address = query_dns_server(server_ip, server_port, domain_name)
48
49      if len(sys.argv) == 5:
50          timeout = sys.argv[4]
51          time.sleep(int(timeout))
52
53      if ip_address == 'Not found':
54          print(f"{domain_name} is not found")
55      elif ip_address == 'Check the server IP and port are correct and check if the server is running':
56          print(ip_address)
57      else:
58          print(f"{domain_name} resolves to: {ip_address[0]} with Response Size: {ip_address[1]}")
59
```

6. In the query_dns_server function, we will create a socket object using AF_INET and SOCK_DGRAM.
7. We will then bind the resolver ip and port with the socket.

```
def query_dns_server(server_ip, server_port, domain):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = (server_ip, server_port)
```

8. We will send the query to the resolver and wait for a response.
9. After we get a response, we will send the return the response to be printed on thescreen.

```
query = domain.encode()
client_socket.sendto(query, server_address)
response, _ = client_socket.recvfrom(4096)  # Increase buffer size for potential larger responses
response_size = len(response)
ip_address = response.decode().strip()
```

Implement 3 enhancement to thebasic version of resolver and client.

Error Handling(Resolver):

1. In the resolver we are checking if the port argument is provided.
2. We also check if the port is valid.
3. In the run_dns_server, we are check if the DNS server is running and if its hasstopped then we will close the program.

Error Handling(Client):

1. In the client, we are checking if the ip, port and domain is provided, if not then wewill print the syntax and close the program
2. We are also check if the ip and port is valid
3. In the query_dns_server function, we are using the try and except statement to check if the DNS is running and responding, if not then we will close the socket, exitthe program.

Client Multiplexing(Resolver):

1. In the resolver, we are using the select module, which help us in client multiplexing.
2. The select module working is similar to multithreading.
3. Using the select, we can handle multiple clients **simultaneously.**

```
# Use select to handle multiple clients
readable, _, _ = select.select([server_socket], [], [], 0.1)
```

Handling Long Messages:

- First we increase the buffer size to 4096. UDP has a buffer size of 512 bytes. So weare receiving message over 512 bytes.
- if the client socket gets into any error while receiving over 512 bytes. we are usingthe while loop to first close the socket and then create a new socket which will communicate over TCP with buffer size of 4096.

```
query = domain.encode()
client_socket.sendto(query, server_address)
response, _ = client_socket.recvfrom(4096)  # Increase buffer size for potential larger responses
response_size = len(response)
ip_address = response.decode().strip()

# Check for truncated response (TC bit set) and retry over TCP if necessary
while response_size >= 512 and "TC" in ip_address:
    client_socket.close()
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(server_address)
    client_socket.sendall(query)
    response = client_socket.recv(4096)
    response_size = len(response)
    ip_address = response.decode().strip()
```

Syntax To Run Resolver and Client.

Resolver

- python resolver.py <PORT>
- e.g. python resolver.py 5555

Client

- Python Client <IP> <PORT> <DOMAIN> <TIMEOUT(OPTIONAL)>
- e.g. python client 127.0.0.1 5555 example.com 5