

Assignment 9

Implementation of TCP/UDP Socket Programming

NAME: Shirish Manoj Bobde

Reg. No.: 812

Roll No.: ECE/21152

Problem Statement 1

Write a TCP/UDP socket program (in C/C++/Java/Python) to establish a connection between client and server. The server should act as a network device maintaining an ARP table.

Implement ARP request and reply functionality.

Display appropriate messages indicating the ARP request and response. Test your program with multiple clients requesting ARP resolution for different IP addresses.

Codes:

Server:

```
import socket

SERVER_IP = '127.0.0.1'
SERVER_PORT = 12345

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.bind((SERVER_IP, SERVER_PORT))

server_socket.listen(5)
print(f"Server listening on {SERVER_IP}:{SERVER_PORT}")

arp_table = {}

def handle_arp_request(ip_address):
    if ip_address in arp_table:
        return arp_table[ip_address]
    else:
        return "IP address not found in ARP table"

def handle_client_connection(client_socket):
    while True:
        data = client_socket.recv(1024).decode()
        if not data:
            break
```

```

        if data.startswith("ARP_REQUEST"):
            ip_address = data.split()[1]
            mac_address = handle_arp_request(ip_address)
            response = f"ARP_RESPONSE {mac_address}"
            client_socket.send(response.encode())

        client_socket.close()

while True:

    client_socket, client_address = server_socket.accept()
    print(f"Connection from {client_address[0]}:{client_address[1]}")

    client_handler = threading.Thread(target=handle_client_connection,
args=(client_socket,))
    client_handler.start()

```

Client:

```

import socket

def main():
    server_host = "127.0.0.1"
    server_port = 8888

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))

    while True:
        ip_address = input("Enter IP address (type 'exit' to quit): ")
        if ip_address.lower() == 'exit':
            break
        client_socket.send(ip_address.encode())
        mac_address = client_socket.recv(1024).decode()
        print(f"The MAC address corresponding to {ip_address} is {mac_address}")

    client_socket.close()

if __name__ == "__main__":
    main()

```

Output:

The image shows two side-by-side screenshots of a code editor. The left screenshot displays the code for 'server1.py'. It defines a 'handle_client' function that receives data from a client, decodes it to get an IP address, looks up the corresponding MAC address, and sends it back. The 'main' function binds the server to '127.0.0.1' on port 8888 and listens for connections. The terminal output shows the server listening and successfully establishing two connections from '127.0.0.1' at different ports.

```
def handle_client(client_socket, address):
    while True:
        data = client_socket.recv(1024)
        if not data:
            break
        ip_address = data.decode()
        mac_address = get_mac_address(ip_address)
        client_socket.send(mac_address.encode())
        client_socket.close()

def main():
    server_host = "127.0.0.1"
    server_port = 8888

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((server_host, server_port))
    server_socket.listen(5)
```

The right screenshot shows the code for 'client12.py' and its terminal output. The client code imports the 'socket' module. The terminal shows the client being run, entering an IP address (224.0.0.251), receiving the corresponding MAC address (01-00-5e-00-00-fb), and then entering another IP address (192.168.29.255) to receive its MAC address (ff-ff-ff-ff-ff-ff).

```
1 import socket

PS C:\Users\ASUS> python -u "C:\Users\ASUS\Desktop\client12.py"
Enter IP address (type 'exit' to quit): 224.0.0.251
The MAC address corresponding to 224.0.0.251 is 01-00-5e-00-00-fb
Enter IP address (type 'exit' to quit): 192.168.29.255
The MAC address corresponding to 192.168.29.255 is ff-ff-ff-ff-ff-ff
Enter IP address (type 'exit' to quit):
```

Problem Statement 2

Write a TCP/UDP socket program (in C/C++/Java/Python) to establish a connection between client and server. The server should act as a network device maintaining a RARP table mapping MAC addresses to IP addresses. Implement RARP request and reply functionality. Display appropriate messages indicating the RARP request and response. Test your program with multiple clients requesting RARP resolution for different MAC addresses.

Code:

Server

```
import subprocess
import socket
import threading

def fetch_arp_table():
    arp_output = subprocess.check_output(["arp", "-a"], text=True)
    arp_entries = arp_output.split("\n")
    arp_table = {}
    for entry in arp_entries:
        if len(entry.strip()) > 0:
            parts = entry.split()
            if len(parts) == 3:
                arp_table[parts[1]] = parts[0]
```

```

        return arp_table

def get_ip_address(mac_address):
    arp_table = fetch_arp_table()
    return arp_table.get(mac_address, "Unknown")

def handle_client(client_socket, address):
    while True:
        data = client_socket.recv(1024)
        if not data:
            break
        mac_address = data.decode()
        ip_address = get_ip_address(mac_address)
        client_socket.send(ip_address.encode())
    client_socket.close()

def main():
    server_host = "127.0.0.1"
    server_port = 8889

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((server_host, server_port))
    server_socket.listen(5)
    print(f"Server listening on {server_host}:{server_port}")

    while True:
        client_socket, address = server_socket.accept()
        print(f"Connection from {address} has been established!")
        client_handler = threading.Thread(
            target=handle_client, args=(client_socket, address)
        )
        client_handler.start()

if __name__ == "__main__":
    main()

```

Client

```

import socket

def main():
    server_host = "127.0.0.1"

```

```

server_port = 8889

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_host, server_port))

while True:
    mac_address = input("Enter MAC address (type 'exit' to quit): ")
    if mac_address.lower() == "exit":
        break
    client_socket.send(mac_address.encode())
    ip_address = client_socket.recv(1024).decode()
    print(f"The IP address corresponding to {mac_address} is {ip_address}")

client_socket.close()

if __name__ == "__main__":
    main()

```

Output:

The image shows two side-by-side screenshots of a code editor (VS Code) with the Python 3.12.2 64-bit interpreter selected. The left window displays the `server2.py` file, and the right window displays the `client.py` file. Both windows have a terminal pane at the bottom showing the execution output.

Left Window (server2.py):

```

def main():
    server_socket.listen(5)
    print(f"Server listening on {server_host}:{server_port}")

    while True:
        client_socket, address = server_socket.accept()
        print(f"Connection from {address} has been establish")
        client_handler = threading.Thread(
            target=handle_client, args=(client_socket, address)
        )
        client_handler.start()

if __name__ == "__main__":
    main()

```

Terminal Output (Left):

```

PS C:\Users\ASUS> python -u "c:\Users\ASUS\Desktop\server2.py"
Server listening on 127.0.0.1:8889
Connection from ('127.0.0.1', 51714) has been established!

```

Right Window (client.py):

```

import socket

def main():
    server_host = "127.0.0.1"
    server_port = 8889

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))

    while True:
        mac_address = input("Enter MAC address (type 'exit' to quit): ")
        if mac_address.lower() == "exit":
            break
        client_socket.send(mac_address.encode())
        ip_address = client_socket.recv(1024).decode()
        print(f"The IP address corresponding to {mac_address} is {ip_address}")

    client_socket.close()

```

Terminal Output (Right):

```

Enter MAC address (type 'exit' to quit): ff-ff-ff-ff-ff-ff
The IP address corresponding to ff-ff-ff-ff-ff-ff is Unknown
Enter MAC address (type 'exit' to quit): ff-ff-ff-ff-ff-ff
The IP address corresponding to ff-ff-ff-ff-ff-ff is 255.255.255.255
Enter MAC address (type 'exit' to quit): 01-00-5e-00-00-fc
The IP address corresponding to 01-00-5e-00-00-fc is 224.0.0.252
Enter MAC address (type 'exit' to quit): 8c-a3-99-b9-90-22
The IP address corresponding to 8c-a3-99-b9-90-22 is Unknown
Enter MAC address (type 'exit' to quit): 8c-a3-99-b9-90-22
The IP address corresponding to 8c-a3-99-b9-90-22 is 192.168.29.1
Enter MAC address (type 'exit' to quit):

```