****

**NETWORKS LAB**

**EXERCISE 4**

Name: Jayannthan P T

Dept: CSE ‘A’

Roll No.: 205001049

Address Resolution Protocol (ARP).

**Aim:**

Develop a chat application between a client and server using UDP. Update the program to support multiple clients (using fd\_set() and select() functions of C.)

**Algorithm:**

**SERVER**

1. Consider the server as a host or a router.

2. Enter hosts/routers’ IP address and MAC address.

3. Listen for any number of client (for broadcasting purpose).

4. Enter the packet details received from a host or its own packet to send to a destination.

The details are:

1. i. Source IP address
2. ii. Source MAC address
3. iii. Destination IP address
4. iv. 16-bit data

Develop an ARP Request packet which is to be broadcasted to all clients. Query packet should contain

**ARPOperation| SourceMAC | SourceIP | DestinationMAC | DestinationIP**

5. When an ARP Reply is received with the Destination MAC address, send the packet to the corresponding destination.

6. Also check the validity of IP and MAC address.

**CLIENT**

1. Can have any number of clients (depends on the backlog).

2. Enter the clients own IP and MAC.

3. When an ARP Request packet is received check whether the Destination IP is its own IP.

4. If not no reply.

5. If yes respond with ARP Reply packet.

ARPOperation|SourceMAC | SourceIP | DestinationMAC | DestinationIP

6. Then receive the packet from the server and display it.

**Code:**

**Server**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <netinet/in.h>

#include <sys/socket.h>

typedef **char** string[50];

#define REQ 1

#define ACK 2

#define DATA 3

typedef **struct** ARP\_PACKET

{

**int** mode;

    string src\_ip;

    string dest\_ip;

    string src\_mac;

    string dest\_mac;

    string data;

} arp;

arp createARPPacket(**int** mode)

{

    arp packet;

    bzero(&packet, sizeof(packet));

    packet.mode = mode;

    printf("\nEnter the details of packet.\n");

    printf("Source IP\t: ");

    scanf(" %s", packet.src\_ip);

    printf("Source MAC\t: ");

    scanf(" %s", packet.src\_mac);

    printf("Destination IP\t: ");

    scanf(" %s", packet.dest\_ip);

    printf("16 bit data\t: ");

    scanf(" %s", packet.data);

    return packet;

}

**void** printPacket(arp packet)

{

    if (packet.mode == REQ)

        printf("%d|%s|%s|%s|%s\n", packet.mode, packet.src\_mac, packet.src\_ip, "00:00:00:00:00:00", packet.dest\_ip);

    else if (packet.mode == ACK)

        printf("%d|%s|%s|%s|%s\n", packet.mode, packet.src\_mac, packet.src\_ip, packet.dest\_ip, packet.dest\_mac);

    else

        printf("%d|%s|%s|%s|%s|%s\n", packet.mode, packet.src\_mac, packet.src\_ip, packet.dest\_ip, packet.dest\_mac, packet.data);

}

**int** main(**int** argc, **char** \*\*argv)

{

    if (argc < 2)

    {

        fprintf(stderr, "Enter port number as second argument!\n");

        exit(EXIT\_FAILURE);

    }

**int** PORT = atoi(argv[1]);

**struct** sockaddr\_in server, client;

**char** buffer[1024];

**int** client\_sockets[10] = {0}, max, fd, sockfd, newfd, activity;

**int** k, i, len, count;

    fd\_set newfds;

    arp packet, recv\_packet;

    packet = createARPPacket(REQ);

    printf("\nDeveloping ARP Request packet\n");

    printPacket(packet);

    printf("\tThe ARP Request packet is broacasted.\n");

    printf("Waiting for ARP Reply...\n");

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sockfd < 0)

    {

        perror("Unable to open socket.\n");

        exit(EXIT\_FAILURE);

    }

    bzero(&server, sizeof(server));

    server.sin\_family = AF\_INET;

    server.sin\_addr.s\_addr = INADDR\_ANY;

    server.sin\_port = htons(PORT);

    if (bind(sockfd, (**struct** sockaddr \*)&server, sizeof(server)) < 0)

    {

        perror("Bind error occurred.\n");

        exit(EXIT\_FAILURE);

    }

    listen(sockfd, 10);

    len = sizeof(client);

    while (1)

    {

        FD\_ZERO(&newfds); *// Clears socket set.*

        FD\_SET(sockfd, &newfds); *// Add sockfd to socket set.*

        max = sockfd;

        for (i = 0; i < 10; i++)

        {

            fd = client\_sockets[i];

            if (fd > 0)

            {

                FD\_SET(fd, &newfds);

            }

            if (fd > max)

            { *// Store the max valued FD.*

                max = fd;

            }

        }

*// Wait indefinitely till any client pings.*

        activity = select(max + 1, &newfds, NULL, NULL, NULL);

        if (activity < 0)

        {

            perror("Select error occurred.\n");

            exit(EXIT\_FAILURE);

        }

*// if sockfd change => new connection request.*

        if (FD\_ISSET(sockfd, &newfds))

        {

            newfd = accept(sockfd, (**struct** sockaddr \*)&client, &len);

            if (newfd < 0)

            {

                perror("Unable to accept the new connection.\n");

                exit(EXIT\_FAILURE);

            }

            send(newfd, (**void** \*)&packet, sizeof(packet), 0);

*// Add the new client on an empty slot.*

            for (i = 0; i < 10; i++)

            {

                if (client\_sockets[i] == 0)

                {

                    client\_sockets[i] = newfd;

                    break;

                }

            }

        }

*// Broadcast on all established connections*

        for (i = 0; i < 10; i++)

        {

            fd = client\_sockets[i];

            bzero((**void** \*)&recv\_packet, sizeof(recv\_packet));

*// Check for change in FD*

            if (FD\_ISSET(fd, &newfds))

            {

                recv(fd, (**void** \*)&recv\_packet, sizeof(recv\_packet), 0);

*// Check ARP response*

                if (recv\_packet.mode == ACK)

                {

                    printf("\nARP Reply received: \n");

                    printPacket(recv\_packet);

                    strcpy(packet.dest\_mac, recv\_packet.src\_mac);

                    packet.mode = DATA;

                    printf("\nSending the packet to: %s\n", packet.dest\_mac);

                    send(newfd, (**void** \*)&packet, sizeof(packet), 0);

                    printf("Packet sent: \n");

                    printPacket(packet);

                    exit(EXIT\_SUCCESS);

                }

            }

        }

    }

    close(sockfd);

    return 0;

}

**Client**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <netinet/in.h>

#include <sys/socket.h>

typedef **char** string[50];

#define REQ 1

#define ACK 2

#define DATA 3

typedef **struct** ARP\_PACKET{

**int** mode;

    string src\_ip;

    string dest\_ip;

    string src\_mac;

    string dest\_mac;

    string data;

}arp;

arp createARPPacket(**int** mode){

    arp packet;

    bzero(&packet, sizeof(packet));

    packet.mode = mode;

    printf("\nEnter the details of packet.\n");

    printf("Source IP\t: ");

    scanf(" %s", packet.src\_ip);

    printf("Source MAC\t: ");

    scanf(" %s", packet.src\_mac);

    printf("Destination IP\t: ");

    scanf(" %s", packet.dest\_ip);

    printf("16 bit data\t: ");

    scanf(" %s", packet.data);

    return packet;

}

**void** printPacket(arp packet){

    if (packet.mode == REQ)

        printf("%d|%s|%s|%s|%s\n", packet.mode, packet.src\_mac, packet.src\_ip, "00:00:00:00:00:00", packet.dest\_ip);

    else if (packet.mode == ACK)

        printf("%d|%s|%s|%s|%s\n", packet.mode, packet.src\_mac, packet.src\_ip, packet.dest\_ip, packet.dest\_mac);

    else

        printf("%d|%s|%s|%s|%s|%s\n", packet.mode, packet.src\_mac, packet.src\_ip, packet.dest\_ip, packet.dest\_mac, packet.data);

}

**int** main(**int** argc, **char** \*\*argv){

     if (argc < 2){

        fprintf(stderr, "Enter port number as second argument!\n");

        exit(EXIT\_FAILURE);

    }

**int** PORT = atoi(argv[1]);

**struct** sockaddr\_in server, client;

**char** buffer[1024];

**int** sockfd, newfd;

**int** len, i, count, k;

    arp packet, recv\_packet;

    printf("\nEnter the IP Address\t: ");

    scanf("%s", packet.src\_ip);

    printf("\nEnter the MAC Address\t: ");

    scanf("%s", packet.src\_mac);

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if(sockfd < 0){

        perror("Unable to open socket.\n");

    }

    bzero(&server, sizeof(server));

    server.sin\_family = AF\_INET;

    server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    server.sin\_port = htons(PORT);

    connect(sockfd, (**struct** sockaddr\*)&server, sizeof(server));

    len = sizeof(client);

    bzero(&recv\_packet, sizeof(recv\_packet));

    recv(sockfd, (**void**\*)&recv\_packet, sizeof(recv\_packet), 0);

    printf("\nARP Request Received: \n");

    printPacket(recv\_packet);

    if(strcmp(packet.src\_ip, recv\_packet.dest\_ip) == 0){

        printf("\nIP Address matches.\n");

        packet.mode = ACK;

        strcpy(packet.dest\_ip, recv\_packet.src\_ip);

        strcpy(packet.dest\_mac, recv\_packet.src\_mac);

        send(sockfd, (**void**\*)&packet, sizeof(packet), 0);

        printf("\nARP Reply Sent: \n");

        printPacket(packet);

        bzero(&recv\_packet, sizeof(recv\_packet));

        recv(sockfd, (**void**\*)&recv\_packet, sizeof(recv\_packet), 0);

        printf("\nReceived Packet is: \n");

        printPacket(recv\_packet);

    }

    else{

        printf("\nIP Address does not match.\n");

    }

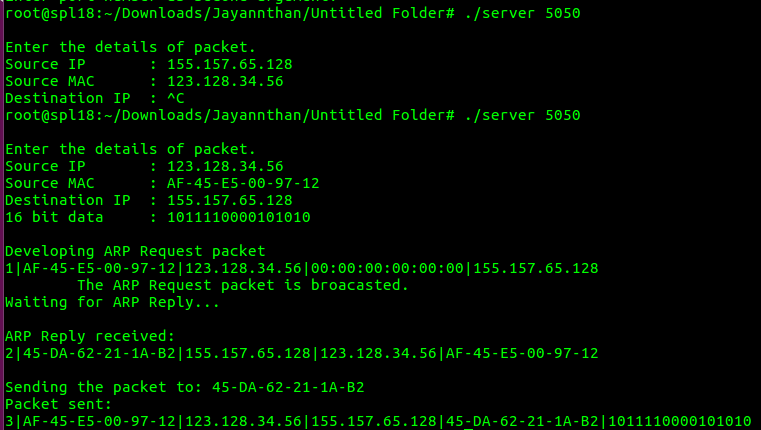
    close(sockfd);

    return 0;

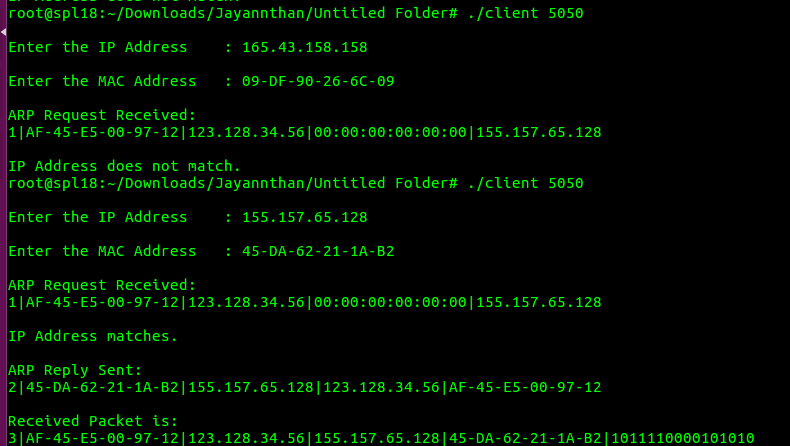
}

**Output:**

Server :



Client:



**Learning outcome:**

Learnt to request and response for ARP packet