SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

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:

i i. Source IP address
ii ii. Source MAC address
iii iii. Destination IP address
iv 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

```c
#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);
```

```c
    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>
```

```c
#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);

}
```

```c
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);
```

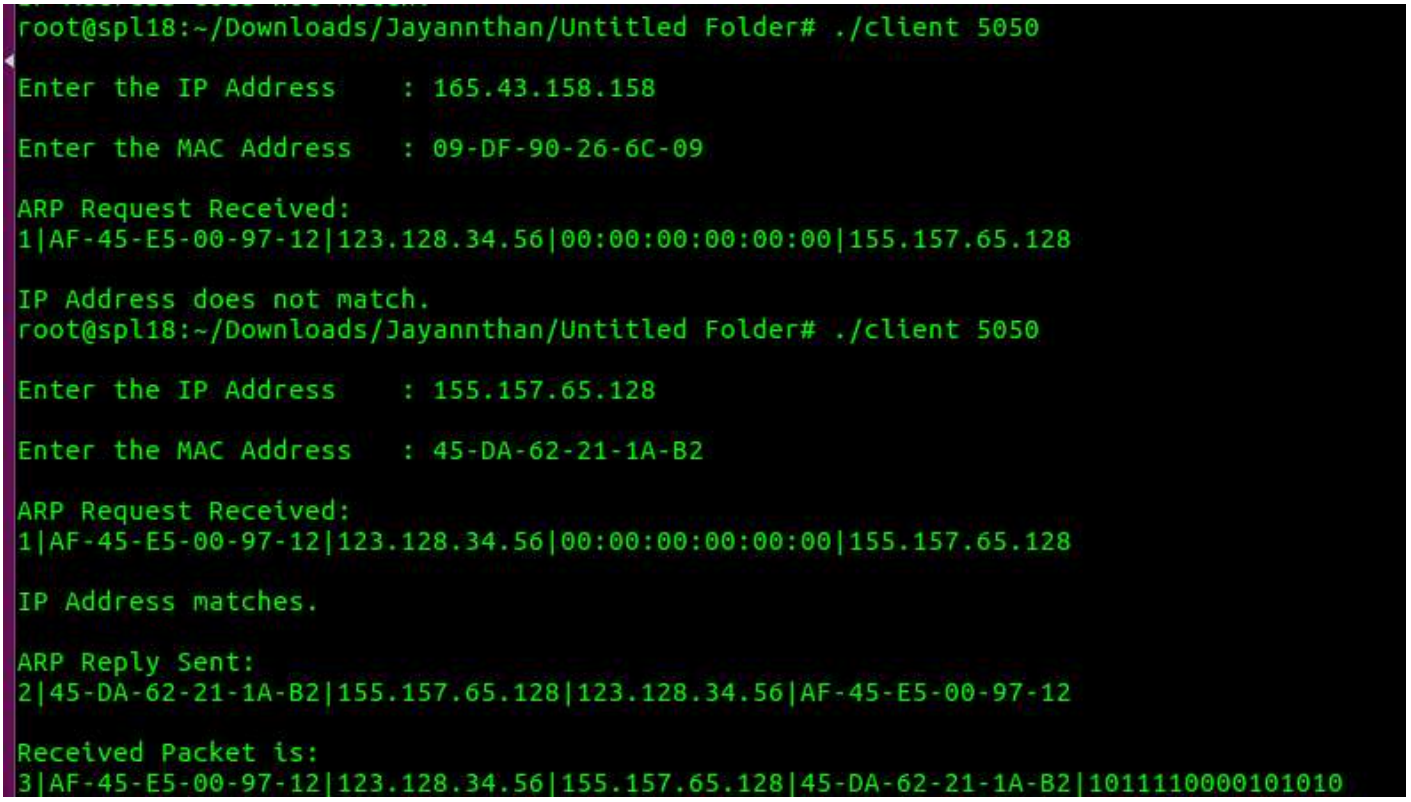
```
        bzero(&recv_packet, sizeof(recv_packet));
        recv(sockfd, (void*)&recv_packet, sizeof(recv_packet), 0);
        printf("\nReceived Packet is: \n");
        printPacket(recv_packet);
    }

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

    close(sockfd);

    return 0;
}
```

Output:

Server :

```
root@spl18:~/Downloads/Jayannthan/Untitled Folder# ./server 5050

Enter the details of packet.
Source IP       : 155.157.65.128
Source MAC      : 123.128.34.56
Destination IP  : ^C
root@spl18:~/Downloads/Jayannthan/Untitled Folder# ./server 5050

Enter the details of packet.
Source IP       : 123.128.34.56
Source MAC      : AF-45-E5-00-97-12
Destination IP  : 155.157.65.128
16 bit data     : 1011110000101010

Developing ARP Request packet
1|AF-45-E5-00-97-12|123.128.34.56|00:00:00:00:00:00|155.157.65.128
        The ARP Request packet is broacasted.
Waiting for ARP Reply...

ARP Reply received:
2|45-DA-62-21-1A-B2|155.157.65.128|123.128.34.56|AF-45-E5-00-97-12

Sending the packet to: 45-DA-62-21-1A-B2
Packet sent:
3|AF-45-E5-00-97-12|123.128.34.56|155.157.65.128|45-DA-62-21-1A-B2|1011110000101010
```

Client:



Learning outcome:

Learnt to request and response for ARP packet