UCS 1511 - Network Lab

Exercise 6 - Address Resolution Protocol

Name:	Mahesh Bharadwaj K
Reg No:	185001089
Semester:	V
Date:	October 10, 2020

---

1 Address Resolution Protocol

Aim:

To simulate ARP using socket programming.

Algorithm

1. Server

1. Create a socket descriptor with **socket()** system call with AF_INET (IPV4 domain), SOCK_STREAM, default protocol and store as sockfd.
2. If sockfd is a negative number, socket creation failed, end program.
3. Create sockaddr_in object to assign IP address and Port number for socket. Set family to AF_INET, IP address to INADDR_ANY to accept connections from any client and port number required.
4. Bind newly created socket to addresss given in sockaddr_in.
5. If bind is non zero, bind failed, print error message and terminate.
6. Listen on the socked defined for as many clients as required. If **listen()** returns non zero value, print error message and terminate.
7. Accept ARP request packet.
8. BEGIN LOOP
    - Detect new connections on socket using **select()** system call.
    - Store the new connection in list of clients.
    - Send the ARP request packet to the new client.
    - Check if client has repsonded to server using **select()** system call.
    - Unpack the received packet and find the MAC address of the client.
    - Send the data packet to the correct client.
9. Close connections on socket using **close()** and terminate program.

2. Client

1. Create a socket descriptor with **socket()** system call with AF_INET (IPV4 domain), SOCK_STREAM, default protocol and store as sockfd.
2. If sockfd is a negative number, socket creation failed, end program.
3. Create sockaddr_in object to assign IP address and Port number for socket. Set family to AF_INET, IP address to localhost(127.0.0.1) to connect to server and port number required.
4. Connect the client to server at address given in socket descriptor using **connect()** system call.
5. If connect() returns -1, connection failed; Print error message and terminate the program.
6. Receive ARP request packet from server.
7. IF IP address of request packet matches client IP address
   - Send acknowledgement packet to server.
   - Receive data packet from server.
   - Display data received.
8. ELSE, IP doesnt match and terminate.
9. Close the connections on socket using **close()** and terminate program.

Program

1. Server Side

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>

#include "ARP.h"

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;
}
```

2. Client Side

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>

#include "ARP.h"

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;
```

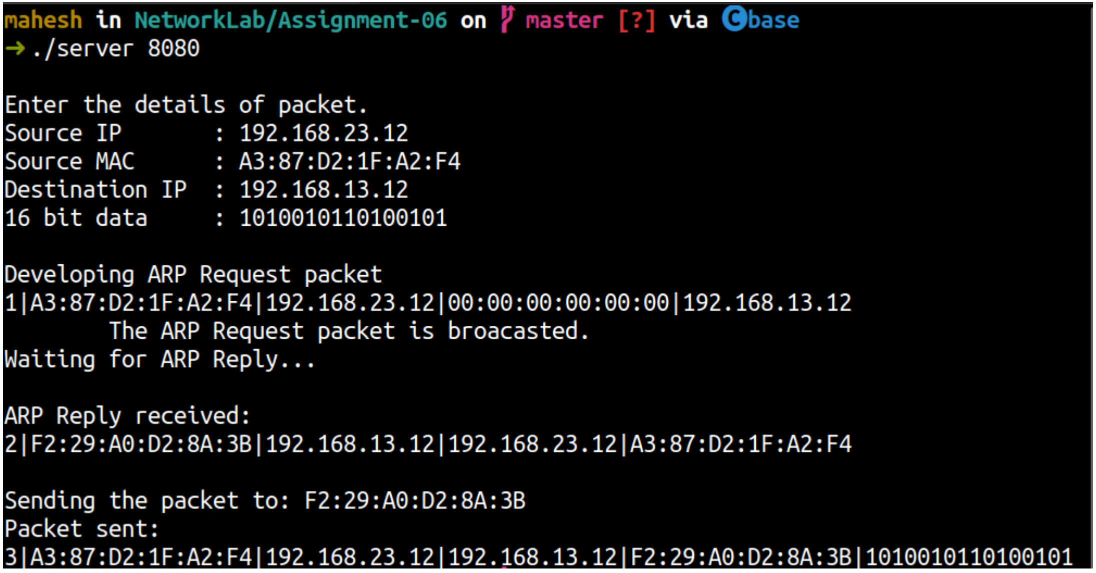
```
        char buffer[1024];
        int sockfd, newfd;
        int len, i, count, k;
        arp packet, recv_packet;

        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;
}
```

3. ARP Specific Functions

```c
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);

}
```

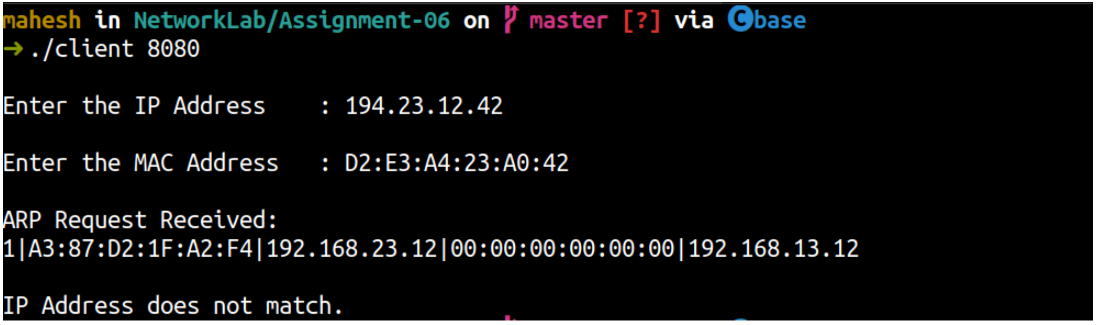
Output

Figure 1: **Server Program Output**



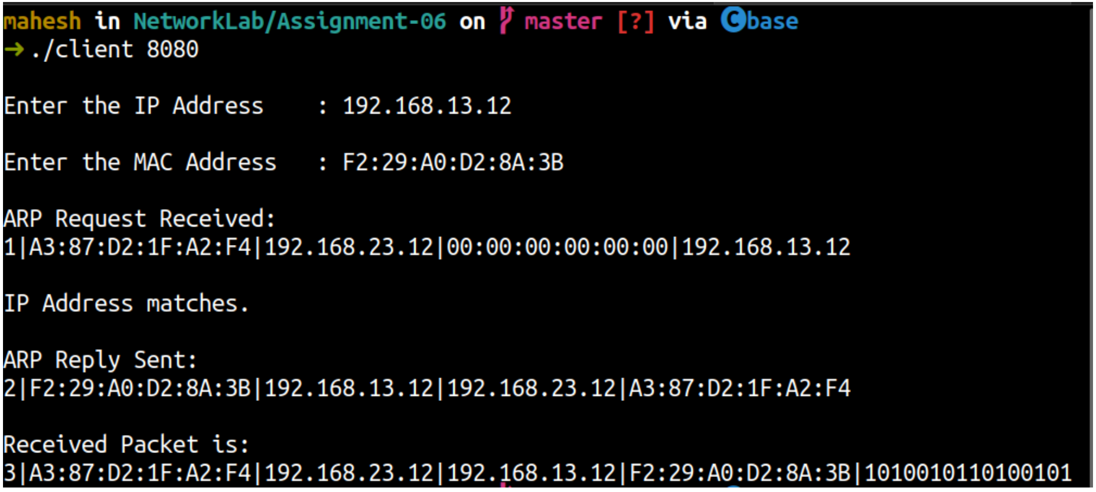
```
mahesh in NetworkLab/Assignment-06 on master [?] via base
➜ ./server 8080

Enter the details of packet.
Source IP       : 192.168.23.12
Source MAC      : A3:87:D2:1F:A2:F4
Destination IP  : 192.168.13.12
16 bit data     : 1010010110100101

Developing ARP Request packet
1|A3:87:D2:1F:A2:F4|192.168.23.12|00:00:00:00:00:00|192.168.13.12
        The ARP Request packet is broacasted.
Waiting for ARP Reply...

ARP Reply received:
2|F2:29:A0:D2:8A:3B|192.168.13.12|192.168.23.12|A3:87:D2:1F:A2:F4

Sending the packet to: F2:29:A0:D2:8A:3B
Packet sent:
3|A3:87:D2:1F:A2:F4|192.168.23.12|192.168.13.12|F2:29:A0:D2:8A:3B|1010010110100101
```

Figure 2: **First Client Program Output**



```
mahesh in NetworkLab/Assignment-06 on master [?] via base
➜ ./client 8080

Enter the IP Address     : 194.23.12.42

Enter the MAC Address    : D2:E3:A4:23:A0:42

ARP Request Received:
1|A3:87:D2:1F:A2:F4|192.168.23.12|00:00:00:00:00:00|192.168.13.12

IP Address does not match.
```

Figure 3: **Second Client Program Output**



```
mahesh in NetworkLab/Assignment-06 on master [?] via base
➜ ./client 8080

Enter the IP Address     : 192.168.13.12

Enter the MAC Address    : F2:29:A0:D2:8A:3B

ARP Request Received:
1|A3:87:D2:1F:A2:F4|192.168.23.12|00:00:00:00:00:00|192.168.13.12

IP Address matches.

ARP Reply Sent:
2|F2:29:A0:D2:8A:3B|192.168.13.12|192.168.23.12|A3:87:D2:1F:A2:F4

Received Packet is:
3|A3:87:D2:1F:A2:F4|192.168.23.12|192.168.13.12|F2:29:A0:D2:8A:3B|1010010110100101
```

Learning Outcomes:

- We learn how to create a simple TCP client server connection.
- We learn how to appropriate system calls to set up Server and Client Programs.
- We learn the structure of ARP packet.
- We handle ARP request and send appropriate response to simulate ARP.