SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
V Semester - CSE 'B'
UCS1511 NETWORKS LAB

Date :08/10/2020 **Name :** Rahul Ram M
Exercise : 06 **Reg No :** 185001121

ADDRESS RESOLUTION PROTOCOL

Learning Objective:

To simulate ARP using socket programming.

Algorithm for Server:

1. Reading the value of source IP, destination IP, Source MAC and data from the user.
2. Concatenating all these to form an ARP request packet.
3. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
4. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
5. Using bind() to binf the socket to the address and port number specified in addr(custom data structure). Here, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.
6. listen() function is used to set the server socket in the passive mode, where it waits for the client to approach the server to make a connection, with maximum number of connection in this case is 5.
7. Intitialize all the values of the client_socket array to 0(means we don’t have to listen to them).
8. Setting a while loop which runs till we terminate using ctrl+z.
   - Clear the list of socket descriptors to monitor using FD_ZERO(&read_fds).
   - Add the descriptor of the server to the list using FD_SET(sockfd, &read_fds).
   - Assign max_sd as server_socket.
   - Using for loop over client_socket array to select valid descriptors , add them to the list of descriptors to monitor and assign the higher number to max_fd.
   - Using select() wait for the activity on one of the sockets in the read_fds indefinetely(timeout is NULL).
   - Using FD_ISSET(server_socket, &read_fds) tests for an incoming connection.
     - If it detects any incoming connection, accept the connection using accept() which creates a socket and assign the new socket any free space in client_socket array.
     - It also sends the ARP packet request to the client using send().
   - Now loop over all the valid fds in the client_socket array

- Using FD_ISSET(sd, &read_fds) tests for any message from the descriptor.
- If the received buffer is empty means the connection is terminated by client with server. We then close the descriptor for that client and make the cleint_sockets[i] to be zero for reuse.
- Else the buffer contains the ARP packet sent by the client to the server.
- Extract the MAC address of the client and print it.
- Now add the Dara to the ARP packet using strcat and send to the client using send().

Algorithm for Client(same for all the clients):

1. Reading the IP address and MAC of the client from the user.
2. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
3. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
4. The above two steps are same as the server.
5. The connect() system call connects the socket referred to by the file descriptor socket_fd to the address specified by server_addr. Server's address and port is specified in server_addr.
6. Recieve the ARP request packet from the server using recv().
7. Extract the source MAC, source IP address, destination IP address from the ARP request packet sent by the server using for loop.
8. Check if the destination IP sent by the server match with IP of this client.
9. If not, print the message and exit.
10. Else, concatenate the MAC address of this client to the ARP packet and send it to the client using send().
11. Recieve the packet from the server that contains the data from the server.

Program for Server:

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

#define PORT 8080

int main()
{
	struct sockaddr_in server_addr, client_addr;
	char buffer[1024];
	char SRC_IP[100], DEST_IP[100], SRC_MAC[100], DEST_MAC[100],
DATA[100], PKT[600];
	int client_sockets[10], max_sd, fd, sockfd, newfd, ping;
	int k, i, count;
```

```
socklen_t len;
fd_set newfds;

printf("\nEnter the details of packet received.\n");
printf("Destination IP\t: ");
scanf("%s", DEST_IP);
printf("Source IP\t: ");
scanf("%s", SRC_IP);
printf("Source MAC\t: ");
scanf("%s", SRC_MAC);
printf("16 bit data\t: ");
scanf("%s", DATA);

printf("\nDeveloping ARP Request packet\n");
strcpy(PKT, "");
strcat(PKT, SRC_MAC);
strcat(PKT, "|");
strcat(PKT, SRC_IP);
strcat(PKT, "|");
strcat(PKT, "00-00-00-00-00-00");
strcat(PKT, "|");
strcat(PKT, DEST_IP);

printf("\t%s\n", PKT);
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");
}

bzero(&server_addr, sizeof(server_addr));

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(PORT);

if(bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
{
        perror("Bind error occurred.\n");
}

listen(sockfd, 5);

for(i = 0; i < 10; i++)
```

```
{
        client_sockets[i] = 0;
}

len = sizeof(client_addr);

while(1)
{
        FD_ZERO(&newfds);                   //Clears socket set.
        FD_SET(sockfd, &newfds);            //Add sockfd to socket set.

        max_sd = sockfd;

        for(i = 0; i < 10; i++){
                fd = client_sockets[i];

                if(fd > 0){
                        FD_SET(fd, &newfds);
                }

                if(fd > max_sd){            //Store the max valued FD.
                        max_sd = fd;
                }
        }


        //Wait indefinitely till any client pings.
        ping = select(max_sd+1, &newfds, NULL, NULL, NULL);

        if(ping < 0){
                perror("Select error occurred.\n");
        }

        //if sockfd change => new connection request.
        if(FD_ISSET(sockfd, &newfds)){
                newfd = accept(sockfd, (struct sockaddr*)&client_addr, &len);

                if(newfd < 0){
                        perror("Unable to accept the new connection.\n");
                }

                strcpy(buffer, PKT);
                send(newfd, buffer, sizeof(buffer), 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;
                    }
                }
            }

            // checking for any response from any connected clients.
            for(i = 0; i < 10; i++)
            {
                fd = client_sockets[i];
                bzero(buffer, sizeof(buffer));

                //Check for change in FD
                if(FD_ISSET(fd, &newfds))
                {
                    recv(fd, buffer, sizeof(buffer), 0);

                    //Check ARP response
                    if(buffer[0])
                    {
                        printf("\nARP Reply received: %s\n", buffer);
                        count = 0;
                        k = 0;
                        for(i = 0; buffer[i]; i++)
                        {
                            if(count == 0)
                            {
                                DEST_MAC[k++] = buffer[i];
                            }
                            if(buffer[i] == '|')
                            {
                                break;
                            }
                        }
                        DEST_MAC[k] = '\0';

                        printf("\nSending the packet to: %s\n", DEST_MAC);
                        bzero(buffer, sizeof(buffer));

                        strcat(buffer, SRC_MAC);
                        strcat(buffer, "|");
                        strcat(buffer, SRC_IP);
                        strcat(buffer, "|");
                        strcat(buffer, DEST_IP);
                        strcat(buffer, "|");
                        strcat(buffer, DEST_MAC);
                        strcat(buffer, "|");
                        strcat(buffer, DATA);
```

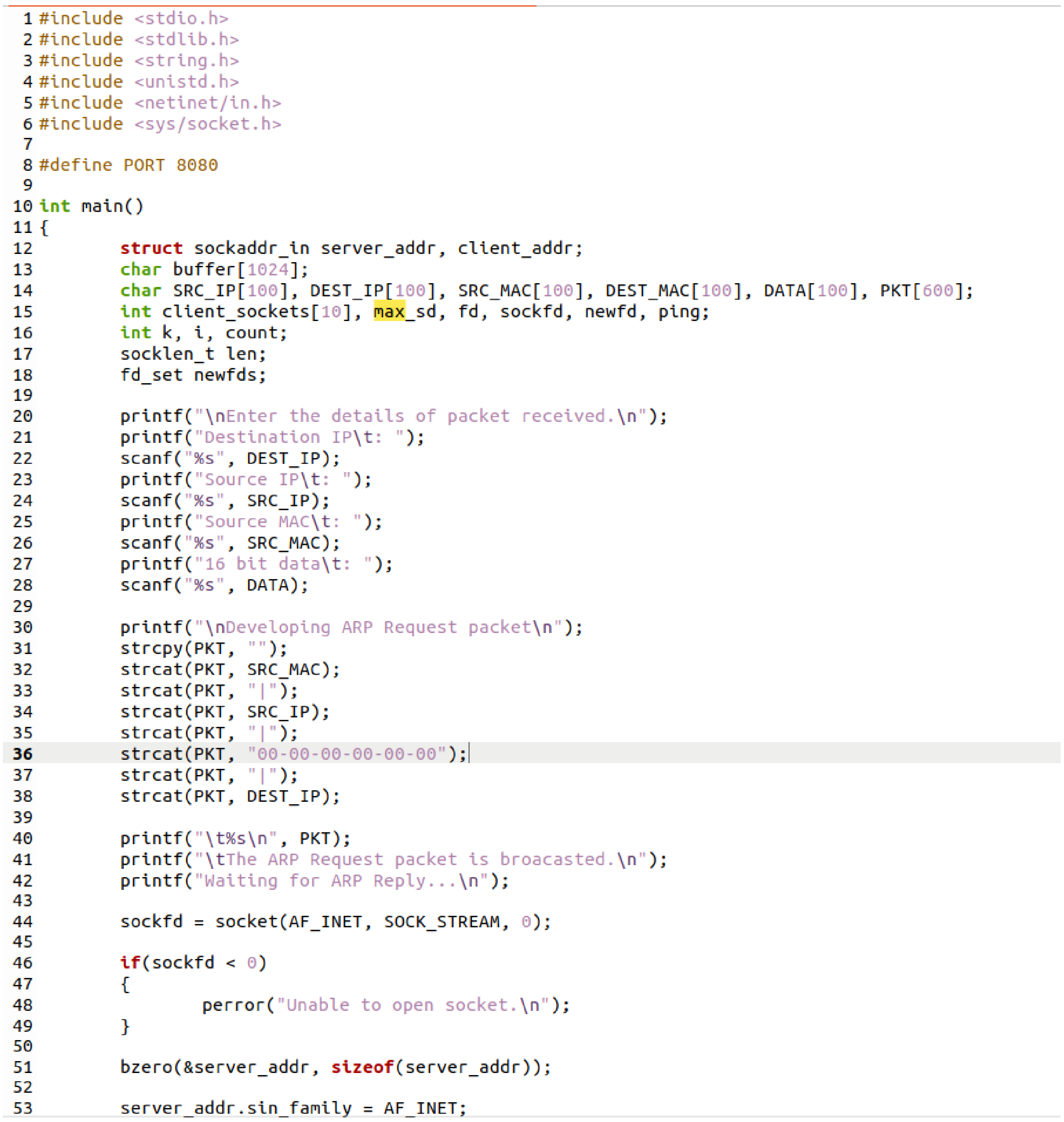
```
					send(newfd, buffer, sizeof(buffer), 0);
					printf("\nPacket Sent: %s\n", buffer);
				}
				else
				{
					close(fd);
					client_sockets[i] = 0;
				}
			}
		}
	}

	return 0;
}
```

Program for Client(same for all the clients):

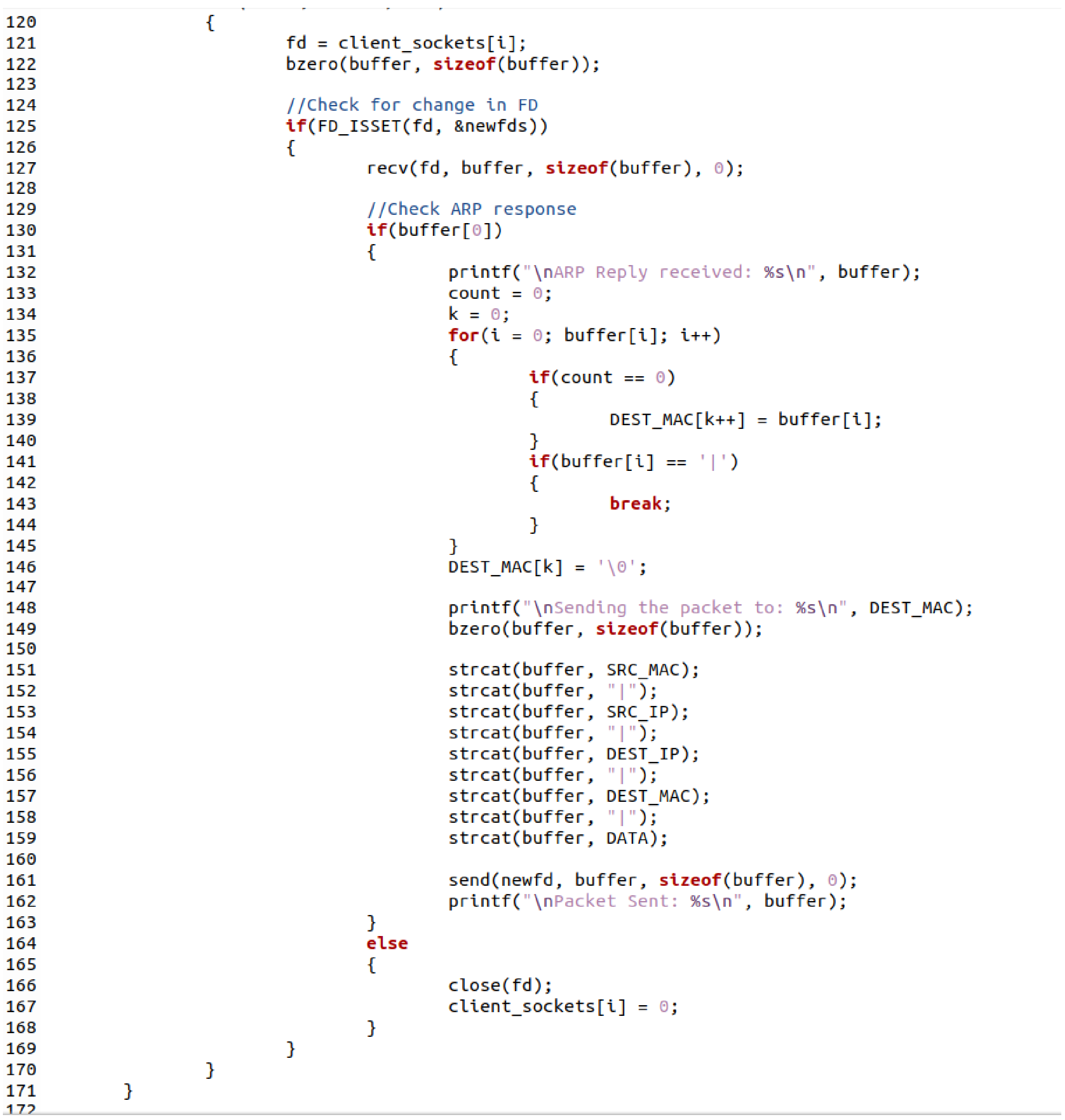
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define PORT 8080

int main()
{
	struct sockaddr_in server_addr;
	char buffer[1024], dest[100], SRC_IP[100], DEST_IP[100], SRC_MAC[100],
DEST_MAC[100];
	int sockfd;
	int i, count, k;

	printf("\nEnter the IP Address\t: ");
	scanf("%s", dest);
	printf("\nEnter the MAC Address\t: ");
	scanf("%s", DEST_MAC);

	sockfd = socket(AF_INET, SOCK_STREAM, 0);

	if(sockfd < 0)
	{
		perror("Unable to open socket.\n");
	}

	bzero(&server_addr, sizeof(server_addr));

	server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
server_addr.sin_port = htons(PORT);

connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));

bzero(buffer, sizeof(buffer));
recv(sockfd, buffer, sizeof(buffer), 0);
printf("\nARP Request Received: %s\n", buffer);

count = 0;
k = 0;
for(i = 0; buffer[i]; i++)
{
	if(buffer[i] == '|')
	{
		if(count == 0)
		{
			SRC_MAC[k] = '\0';
		}
		else if(count == 1)
		{
			SRC_IP[k] = '\0';
		}
		else if(count == 3)
		{
			DEST_IP[k] = '\0';
		}
		count++;
		k = 0;
	}
	else if(count == 0)
	{
		SRC_MAC[k++] = buffer[i];
	}
	else if(count == 1)
	{
		SRC_IP[k++] = buffer[i];
	}
	else if(count == 3)
	{
		DEST_IP[k++] = buffer[i];
	}

}

if(strcmp(dest, DEST_IP) == 0)
{
	bzero(buffer, sizeof(buffer));
```

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

                strcat(buffer, DEST_MAC);
                strcat(buffer, "|");
                strcat(buffer, DEST_IP);
                strcat(buffer, "|");
                strcat(buffer, SRC_IP);
                strcat(buffer, "|");
                strcat(buffer, SRC_MAC);
                send(sockfd, buffer, sizeof(buffer), 0);
                printf("\nARP Reply Sent: %s\n", buffer);

                bzero(buffer, sizeof(buffer));
                recv(sockfd, buffer, sizeof(buffer), 0);
                printf("\nReceived Packet is: %s\n", buffer);
        }

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

        close(sockfd);

        return 0;
}
```

Screenshot for Server:

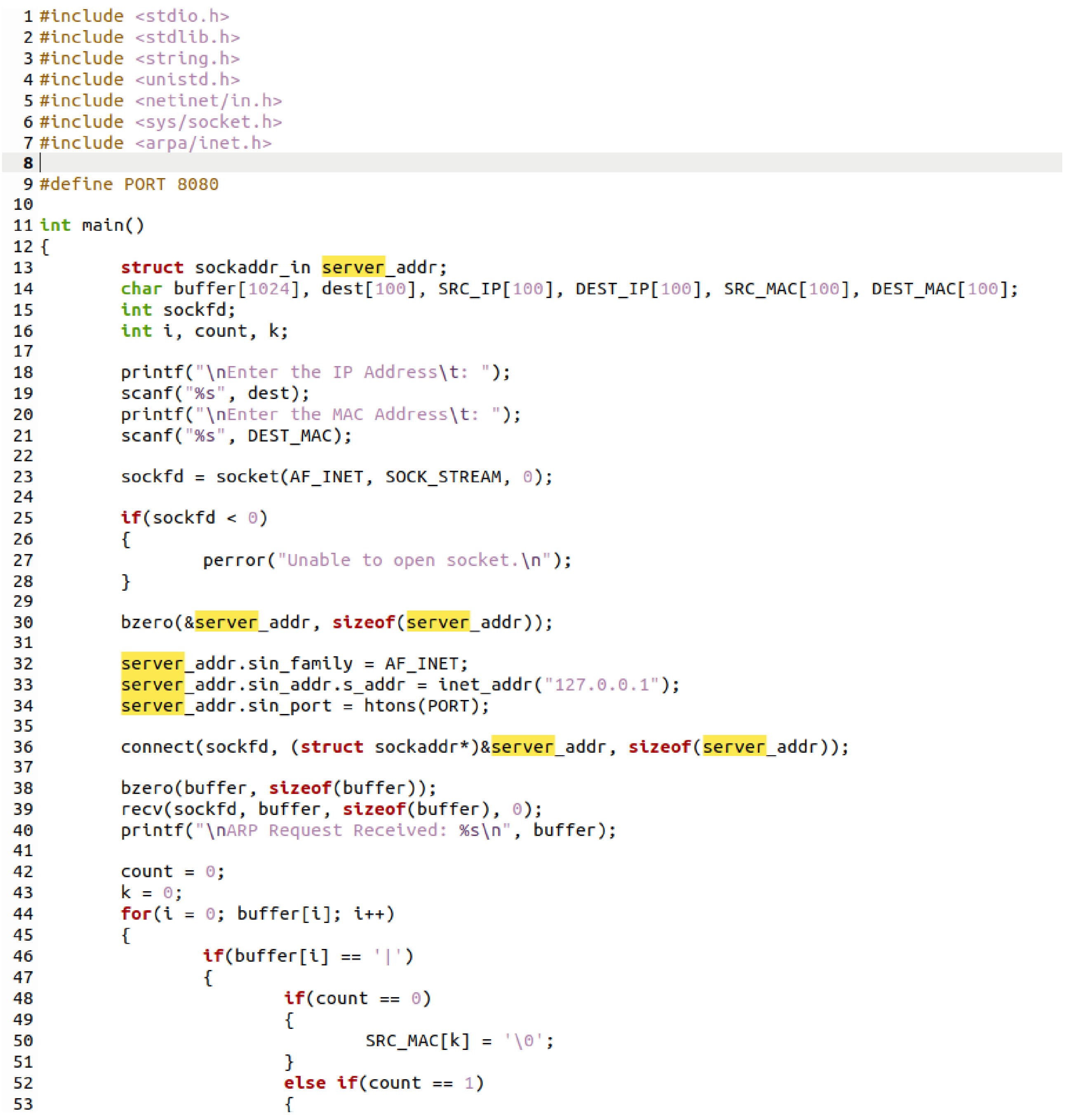


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 8080

int main()
{
        struct sockaddr_in server_addr, client_addr;
        char buffer[1024];
        char SRC_IP[100], DEST_IP[100], SRC_MAC[100], DEST_MAC[100], DATA[100], PKT[600];
        int client_sockets[10], max_sd, fd, sockfd, newfd, ping;
        int k, i, count;
        socklen_t len;
        fd_set newfds;

        printf("\nEnter the details of packet received.\n");
        printf("Destination IP\t: ");
        scanf("%s", DEST_IP);
        printf("Source IP\t: ");
        scanf("%s", SRC_IP);
        printf("Source MAC\t: ");
        scanf("%s", SRC_MAC);
        printf("16 bit data\t: ");
        scanf("%s", DATA);

        printf("\nDeveloping ARP Request packet\n");
        strcpy(PKT, "");
        strcat(PKT, SRC_MAC);
        strcat(PKT, "|");
        strcat(PKT, SRC_IP);
        strcat(PKT, "|");
        strcat(PKT, "00-00-00-00-00-00");
        strcat(PKT, "|");
        strcat(PKT, DEST_IP);

        printf("\t%s\n", PKT);
        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");
        }

        bzero(&server_addr, sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(PORT);

        if(bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Bind error occurred.\n");
        }

        listen(sockfd, 5);

        for(i = 0; i < 10; i++)
        {
                client_sockets[i] = 0;
        }

        len = sizeof(client_addr);

        while(1)
        {
                FD_ZERO(&newfds);               //Clears socket set.
                FD_SET(sockfd, &newfds);        //Add sockfd to socket set.

                max_sd = sockfd;

                for(i = 0; i < 10; i++){
                        fd = client_sockets[i];

                        if(fd > 0){
                                FD_SET(fd, &newfds);
                        }

                        if(fd > max_sd){                //Store the max valued FD.
                                max_sd = fd;
                        }
                }


                //Wait indefinitely till any client pings.
                ping = select(max_sd+1, &newfds, NULL, NULL, NULL);

                if(ping < 0){
                        perror("Select error occurred.\n");
                }

                //if sockfd change => new connection request.
                if(FD_ISSET(sockfd, &newfds)){
                        newfd = accept(sockfd, (struct sockaddr*)&client_addr, &len);

                        if(newfd < 0){
                                perror("Unable to accept the new connection.\n");
                        }

                        strcpy(buffer, PKT);
                        send(newfd, buffer, sizeof(buffer), 0);
```

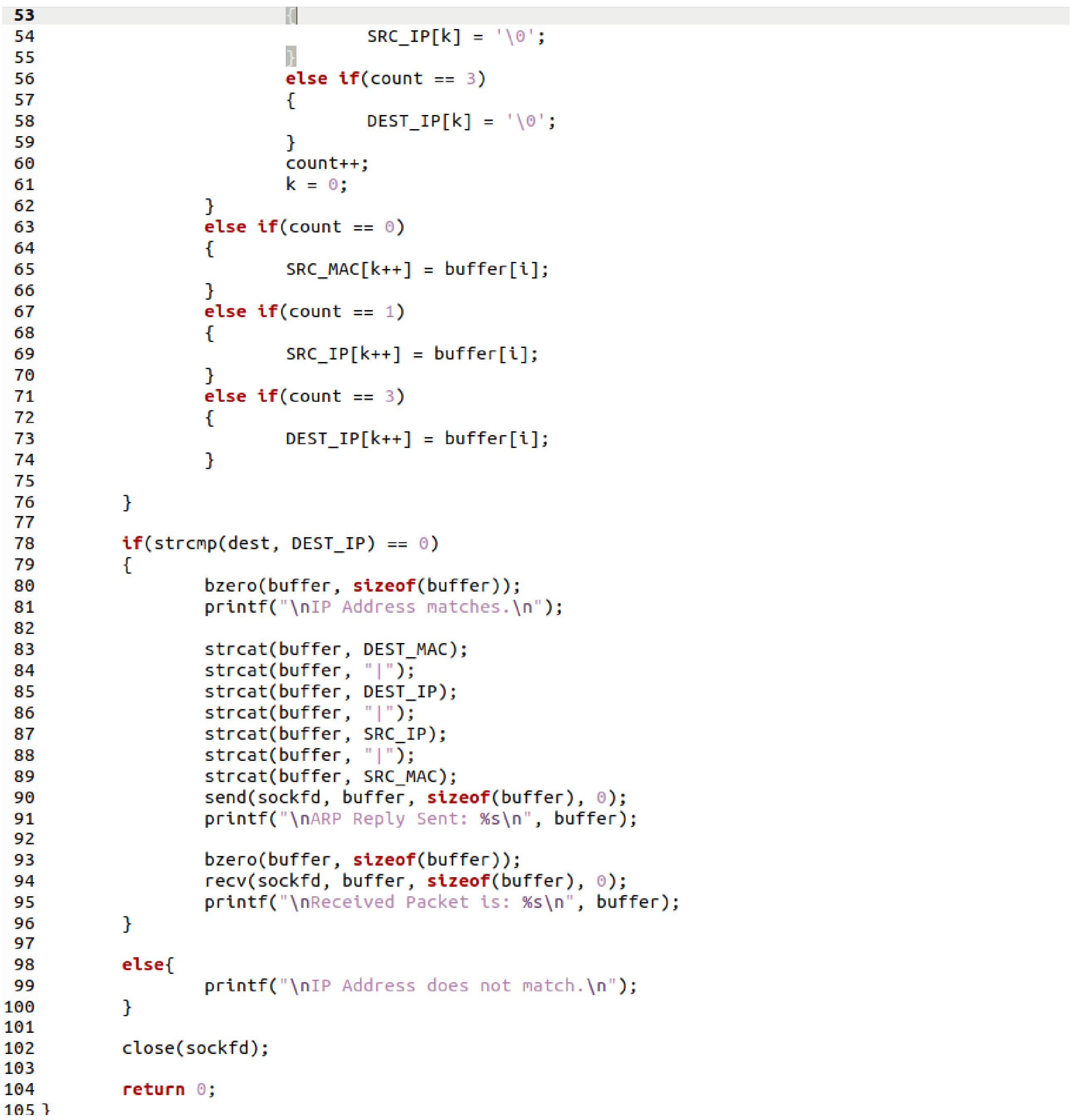


```
                {
                        fd = client_sockets[i];
                        bzero(buffer, sizeof(buffer));

                        //Check for change in FD
                        if(FD_ISSET(fd, &newfds))
                        {
                                recv(fd, buffer, sizeof(buffer), 0);

                                //Check ARP response
                                if(buffer[0])
                                {
                                        printf("\nARP Reply received: %s\n", buffer);
                                        count = 0;
                                        k = 0;
                                        for(i = 0; buffer[i]; i++)
                                        {
                                                if(count == 0)
                                                {
                                                        DEST_MAC[k++] = buffer[i];
                                                }
                                                if(buffer[i] == '|')
                                                {
                                                        break;
                                                }
                                        }
                                        DEST_MAC[k] = '\0';

                                        printf("\nSending the packet to: %s\n", DEST_MAC);
                                        bzero(buffer, sizeof(buffer));

                                        strcat(buffer, SRC_MAC);
                                        strcat(buffer, "|");
                                        strcat(buffer, SRC_IP);
                                        strcat(buffer, "|");
                                        strcat(buffer, DEST_IP);
                                        strcat(buffer, "|");
                                        strcat(buffer, DEST_MAC);
                                        strcat(buffer, "|");
                                        strcat(buffer, DATA);

                                        send(newfd, buffer, sizeof(buffer), 0);
                                        printf("\nPacket Sent: %s\n", buffer);
                                }
                                else
                                {
                                        close(fd);
                                        client_sockets[i] = 0;
                                }
                        }
                }
        }
```

Screenshot for Client:

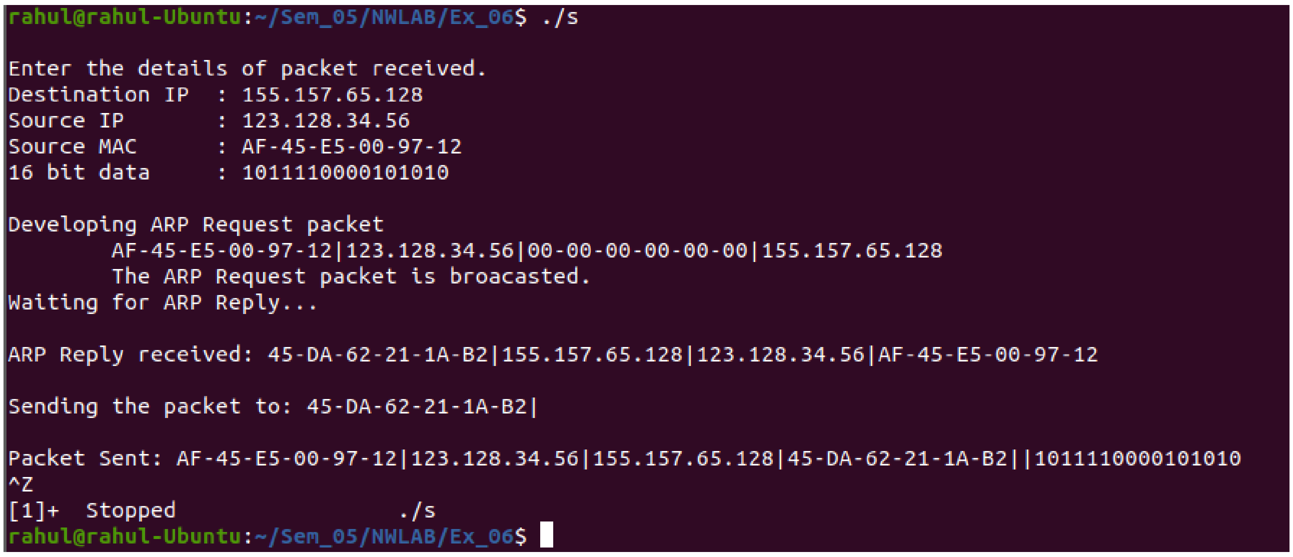


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define PORT 8080

int main()
{
        struct sockaddr_in server_addr;
        char buffer[1024], dest[100], SRC_IP[100], DEST_IP[100], SRC_MAC[100], DEST_MAC[100];
        int sockfd;
        int i, count, k;

        printf("\nEnter the IP Address\t: ");
        scanf("%s", dest);
        printf("\nEnter the MAC Address\t: ");
        scanf("%s", DEST_MAC);

        sockfd = socket(AF_INET, SOCK_STREAM, 0);

        if(sockfd < 0)
        {
                perror("Unable to open socket.\n");
        }

        bzero(&server_addr, sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(PORT);

        connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));

        bzero(buffer, sizeof(buffer));
        recv(sockfd, buffer, sizeof(buffer), 0);
        printf("\nARP Request Received: %s\n", buffer);

        count = 0;
        k = 0;
        for(i = 0; buffer[i]; i++)
        {
                if(buffer[i] == '|')
                {
                        if(count == 0)
                        {
                                SRC_MAC[k] = '\0';
                        }
                        else if(count == 1)
                        {
```



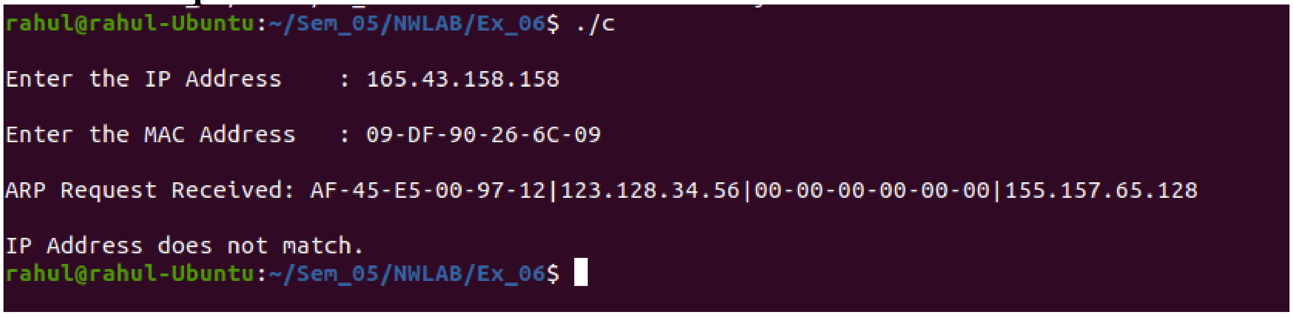
```
                        {
                                SRC_IP[k] = '\0';
                        }
                        else if(count == 3)
                        {
                                DEST_IP[k] = '\0';
                        }
                        count++;
                        k = 0;
                }
                else if(count == 0)
                {
                        SRC_MAC[k++] = buffer[i];
                }
                else if(count == 1)
                {
                        SRC_IP[k++] = buffer[i];
                }
                else if(count == 3)
                {
                        DEST_IP[k++] = buffer[i];
                }

        }

        if(strcmp(dest, DEST_IP) == 0)
        {
                bzero(buffer, sizeof(buffer));
                printf("\nIP Address matches.\n");

                strcat(buffer, DEST_MAC);
                strcat(buffer, "|");
                strcat(buffer, DEST_IP);
                strcat(buffer, "|");
                strcat(buffer, SRC_IP);
                strcat(buffer, "|");
                strcat(buffer, SRC_MAC);
                send(sockfd, buffer, sizeof(buffer), 0);
                printf("\nARP Reply Sent: %s\n", buffer);

                bzero(buffer, sizeof(buffer));
                recv(sockfd, buffer, sizeof(buffer), 0);
                printf("\nReceived Packet is: %s\n", buffer);
        }

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

        close(sockfd);

        return 0;
}
```

Server Output:



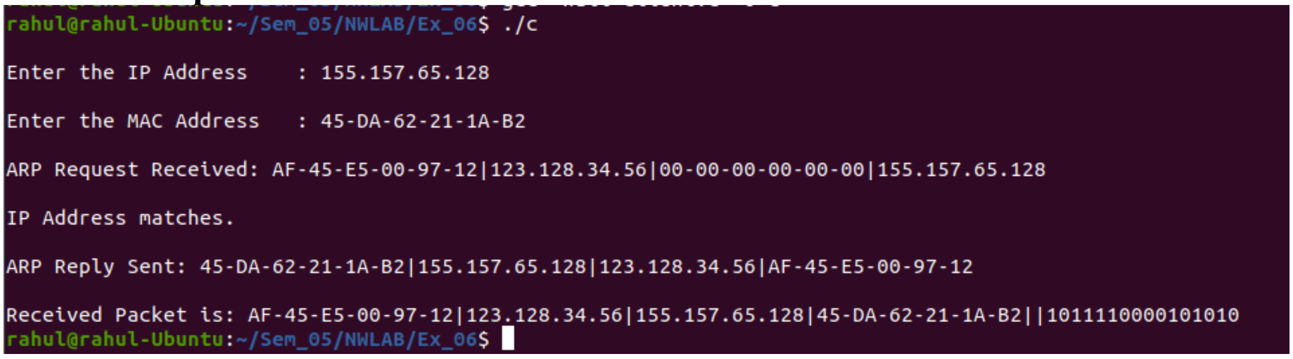
```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ ./s

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

Developing ARP Request packet
        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: 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: AF-45-E5-00-97-12|123.128.34.56|155.157.65.128|45-DA-62-21-1A-B2||1011110000101010
^Z
[1]+  Stopped                 ./s
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ 
```

Client1 Output:



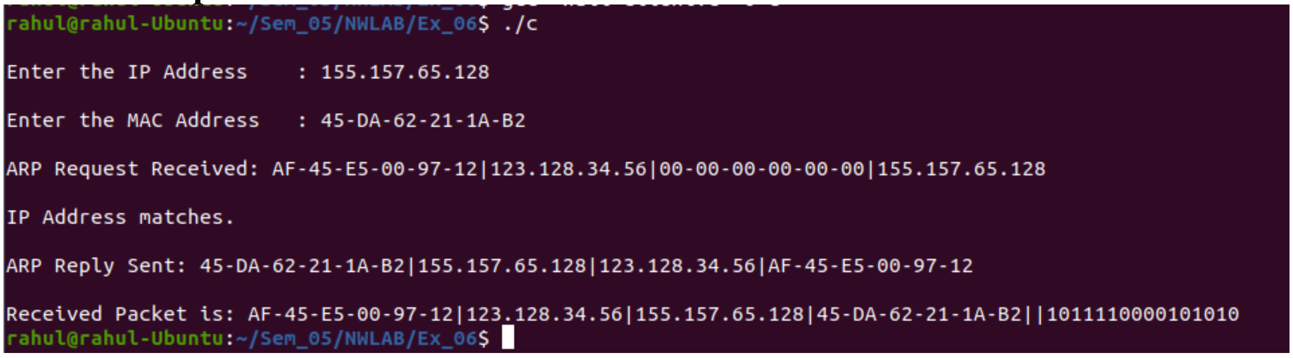
```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ ./c

Enter the IP Address    : 165.43.158.158

Enter the MAC Address   : 09-DF-90-26-6C-09

ARP Request Received: AF-45-E5-00-97-12|123.128.34.56|00-00-00-00-00-00|155.157.65.128

IP Address does not match.
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ 
```

Client2 Output:



```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ ./c

Enter the IP Address    : 155.157.65.128

Enter the MAC Address   : 45-DA-62-21-1A-B2

ARP Request Received: AF-45-E5-00-97-12|123.128.34.56|00-00-00-00-00-00|155.157.65.128

IP Address matches.

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

Received Packet is: AF-45-E5-00-97-12|123.128.34.56|155.157.65.128|45-DA-62-21-1A-B2||1011110000101010
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ 
```

Client3 Output:



```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ ./c

Enter the IP Address    : 155.157.65.128

Enter the MAC Address   : 45-DA-62-21-1A-B2

ARP Request Received: AF-45-E5-00-97-12|123.128.34.56|00-00-00-00-00-00|155.157.65.128

IP Address matches.

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

Received Packet is: AF-45-E5-00-97-12|123.128.34.56|155.157.65.128|45-DA-62-21-1A-B2||1011110000101010
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_06$ 
```

Learning Outcomes:

This assignment helped me to

1. Write program for server and client with socket programming.
2. Understand various functions invloved in creating, estabilishing, maintaining, Sending, recieving and termininating the connection between the server and client.
3. Connect multiple clients to the server using select() system call.
4. Understand various functions of ARP protocol.
5. Simulate the functions of ARP using socket programming.