**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**UCS1511 NETWORKS LAB**

# Exercise 6 - Address Resolution Protocol

**Name :** Kshitij Sharma **Roll No. :** 185001080 **Date :** 06/10/2020

**1 Address Resolution Protocol**

# Aim:

To simulate ARP using socket programming.

# Algorithm

## 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.

## 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

## 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;

}

## 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;

}

## ARP Specific Functions

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,

*‹→*

else

packet.dest\_ip, packet.dest\_mac);

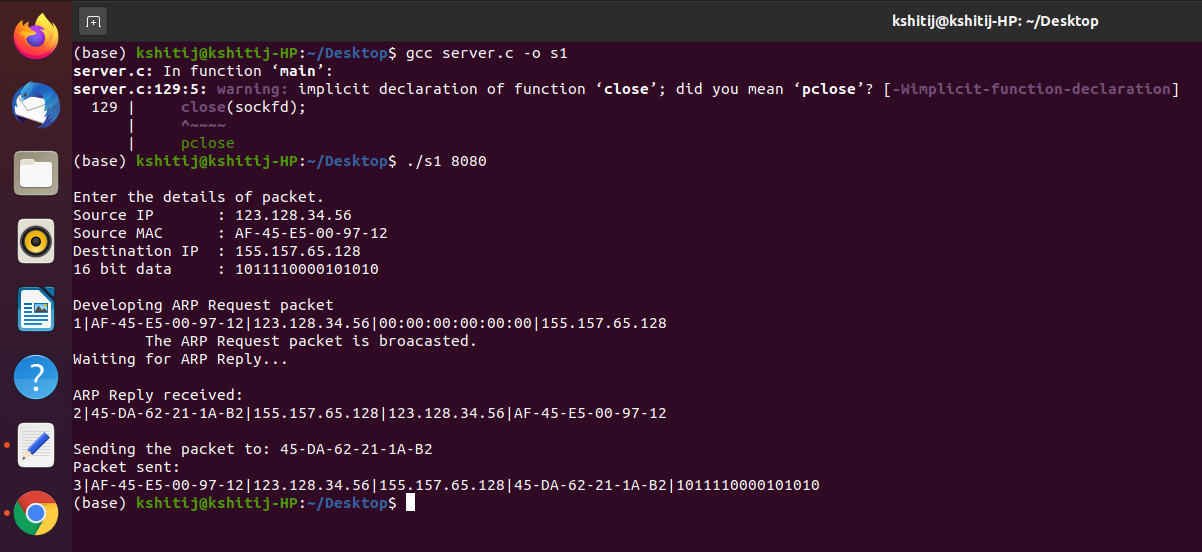
printf(" d| s| s| s| s| s\n", packet.mode, packet.src\_mac, packet.src\_ip,

*‹→* packet.dest\_ip, packet.dest\_mac, packet.data);

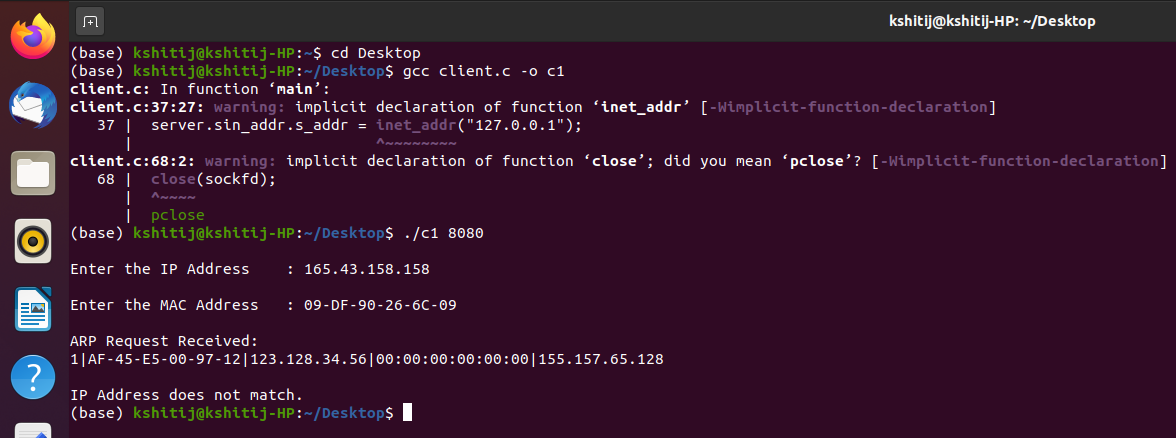
}

# Output

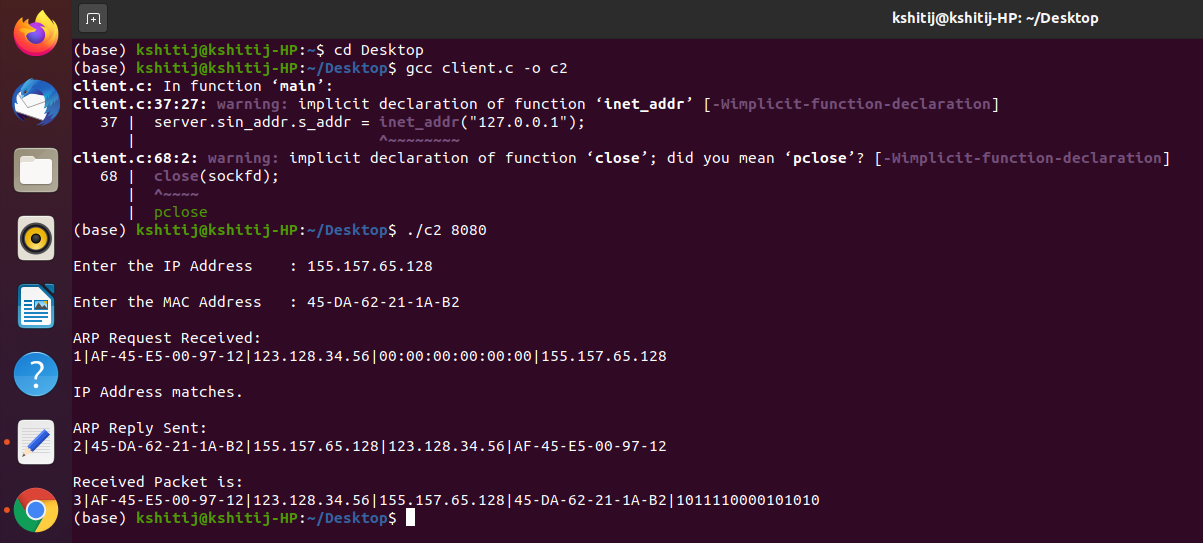
**Server:**



**Client 1:**



**Client 2:**



# 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.