Gita Alekhya Paul 25/09/2021

Computer Networks
RA1911030010014
Experiment - 10
ARP Implementation Using UDP

Aim: To create an ARP Implementation Using UDP.

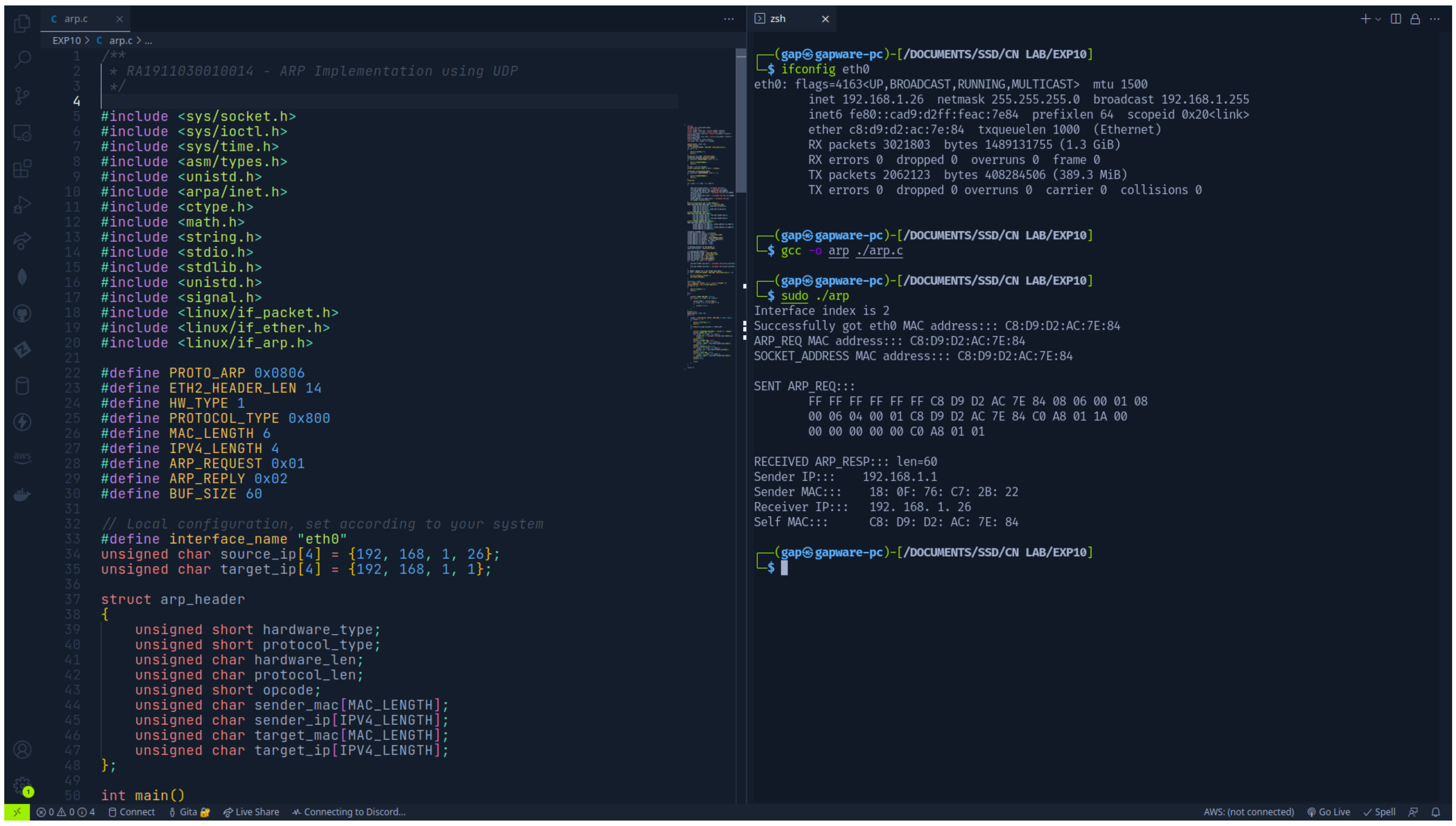
ARP Implementation Code:

```
/**
* RA1911030010014 - ARP Implementation using UDP
*/

#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <asm/types.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <ctype.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>

#define PROTO_ARP 0x0806
#define ETH2_HEADER_LEN 14
#define HW_TYPE 1
#define PROTOCOL_TYPE 0x800
#define MAC_LENGTH 6
#define IPV4_LENGTH 4
#define ARP_REQUEST 0x01
#define ARP_REPLY 0x02
#define BUF_SIZE 60

// Local configuration, set according to your system
#define interface_name "eth0"
unsigned char source_ip[4] = {192, 168, 1, 26};
unsigned char target_ip[4] = {192, 168, 1, 1};

struct arp_header
{
   unsigned short hardware_type;
   unsigned short protocol_type;
   unsigned char hardware_len;
   unsigned char protocol_len;
   unsigned short opcode;
   unsigned char sender_mac[MAC_LENGTH];
   unsigned char sender_ip[IPV4_LENGTH];
   unsigned char target_mac[MAC_LENGTH];
   unsigned char target_ip[IPV4_LENGTH];
};

int main()
{
   int sd;
   unsigned char buffer[BUF_SIZE];
   struct ifreq ifr;
   struct ethhdr *send_req = (struct ethhdr *)buffer;
```

```c
    struct ethhdr *rcv_resp = (struct ethhdr *)buffer;
    struct arp_header *arp_req = (struct arp_header *)(buffer + ETH2_HEADER_LEN);
    struct arp_header *arp_resp = (struct arp_header *)(buffer + ETH2_HEADER_LEN);
    struct sockaddr_ll socket_address;
    int index, ret, length = 0, ifindex;

    memset(buffer, 0x00, 60);
    /*open socket*/
    sd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    if (sd == -1)
    {
        perror("socket():");
        exit(1);
    }
    strcpy(ifr.ifr_name, interface_name);
    /*retrieve ethernet interface index*/
    if (ioctl(sd, SIOCGIFINDEX, &ifr) == -1)
    {
        perror("SIOCGIFINDEX");
        exit(1);
    }
    ifindex = ifr.ifr_ifindex;
    printf("Interface index is %d\n", ifindex);

    /*retrieve corresponding MAC*/
    if (ioctl(sd, SIOCGIFHWADDR, &ifr) == -1)
    {
        perror("SIOCGIFINDEX");
        exit(1);
    }
    close(sd);

    for (index = 0; index < 6; index++)
    {

        send_req->h_dest[index] = (unsigned char)0xff;
        arp_req->target_mac[index] = (unsigned char)0x00;
        /* Filling the source  mac address in the header*/
        send_req->h_source[index] = (unsigned char)ifr.ifr_hwaddr.sa_data[index];
        arp_req->sender_mac[index] = (unsigned char)ifr.ifr_hwaddr.sa_data[index];
        socket_address.sll_addr[index] = (unsigned char)ifr.ifr_hwaddr.sa_data[index];
    }
    printf("Successfully got %s MAC address::: %02X:%02X:%02X:%02X:%02X:%02X\n", interface_name,
           send_req->h_source[0], send_req->h_source[1], send_req->h_source[2],
           send_req->h_source[3], send_req->h_source[4], send_req->h_source[5]);
    printf("ARP_REQ MAC address::: %02X:%02X:%02X:%02X:%02X:%02X\n",
           arp_req->sender_mac[0], arp_req->sender_mac[1], arp_req->sender_mac[2],
           arp_req->sender_mac[3], arp_req->sender_mac[4], arp_req->sender_mac[5]);
    printf("SOCKET_ADDRESS MAC address::: %02X:%02X:%02X:%02X:%02X:%02X\n",
           socket_address.sll_addr[0], socket_address.sll_addr[1], socket_address.sll_addr[2],
           socket_address.sll_addr[3], socket_address.sll_addr[4], socket_address.sll_addr[5]);

    /*prepare sockaddr_ll*/
    socket_address.sll_family = AF_PACKET;
    socket_address.sll_protocol = htons(ETH_P_ARP);
    socket_address.sll_ifindex = ifindex;
    socket_address.sll_hatype = htons(ARPHRD_ETHER);
    socket_address.sll_pkttype = (PACKET_BROADCAST);
    socket_address.sll_halen = MAC_LENGTH;
    socket_address.sll_addr[6] = 0x00;
    socket_address.sll_addr[7] = 0x00;

    /* Setting protocol of the packet */
    send_req->h_proto = htons(ETH_P_ARP);
    /* Creating ARP request */
    arp_req->hardware_type = htons(HW_TYPE);
    arp_req->protocol_type = htons(ETH_P_IP);
    arp_req->hardware_len = MAC_LENGTH;
    arp_req->protocol_len = IPV4_LENGTH;
    arp_req->opcode = htons(ARP_REQUEST);
    for (index = 0; index < 5; index++)
    {
        arp_req->sender_ip[index] = (unsigned char)source_ip[index];
        arp_req->target_ip[index] = (unsigned char)target_ip[index];
    }
    // Submit request for a raw socket descriptor.
    if ((sd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) < 0)
    {
        perror("socket() failed ");
        exit(EXIT_FAILURE);
    }

    buffer[32] = 0x00;
    ret = sendto(sd, buffer, 42, 0, (struct sockaddr *)&socket_address, sizeof(socket_address));
    if (ret == -1)
    {
        perror("sendto():");
        exit(1);
    }
    else
    {
        printf("\nSENT ARP_REQ:::\n\t");
        for (index = 0; index < 42; index++)
        {
            printf("%02X ", buffer[index]);
            if (index % 16 == 0 && index != 0)
            {
                printf("\n\t");
            }
        }
    }
    printf("\n");
    memset(buffer, 0x00, 60);
    while (1)
    {
        length = recvfrom(sd, buffer, BUF_SIZE, 0, NULL, NULL);
        if (length == -1)
        {
            perror("recvfrom():");
            exit(1);
        }
        if (htons(rcv_resp->h_proto) == PROTO_ARP)
        {

            printf("\nRECEIVED ARP_RESP::: len=%d \n", length);
            printf("Sender IP:::\t");
            for (index = 0; index < 4; index++)
                printf("%u.", (unsigned int)arp_resp->sender_ip[index]);
            printf("\b ");
            printf("\nSender MAC:::\t");
            for (index = 0; index < 6; index++)
                printf(" %02X:", arp_resp->sender_mac[index]);
            printf("\b ");
            printf("\nReceiver IP:::\t");
            for (index = 0; index < 4; index++)
                printf(" %u.", arp_resp->target_ip[index]);
            printf("\b ");
            printf("\nSelf MAC:::\t");
            for (index = 0; index < 6; index++)
```

```c
                printf(" %02X:", arp_resp->target_mac[index]);
            printf("\b ");
            printf("\n");

            break;
        }
    }

    return 0;
}
```

Output:



arp.c

EXP10 > arp.c > ...

```c
/**
 * RA1911030010014 - ARP Implementation using UDP
 */

#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <asm/types.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <ctype.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>

#define PROTO_ARP 0x0806
#define ETH2_HEADER_LEN 14
#define HW_TYPE 1
#define PROTOCOL_TYPE 0x800
#define MAC_LENGTH 6
#define IPV4_LENGTH 4
#define ARP_REQUEST 0x01
#define ARP_REPLY 0x02
#define BUF_SIZE 60

// Local configuration, set according to your system
#define interface_name "eth0"
unsigned char source_ip[4] = {192, 168, 1, 26};
unsigned char target_ip[4] = {192, 168, 1, 1};

struct arp_header
{
    unsigned short hardware_type;
    unsigned short protocol_type;
    unsigned char hardware_len;
    unsigned char protocol_len;
    unsigned short opcode;
    unsigned char sender_mac[MAC_LENGTH];
    unsigned char sender_ip[IPV4_LENGTH];
    unsigned char target_mac[MAC_LENGTH];
    unsigned char target_ip[IPV4_LENGTH];
};

int main()
```

zsh

```
┌──(gap㉿gapware-pc)-[/DOCUMENTS/SSD/CN LAB/EXP10]
└─$ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.26  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::cad9:d2ff:feac:7e84  prefixlen 64  scopeid 0x20<link>
        ether c8:d9:d2:ac:7e:84  txqueuelen 1000  (Ethernet)
        RX packets 3021803  bytes 1489131755 (1.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2062123  bytes 408284506 (389.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


┌──(gap㉿gapware-pc)-[/DOCUMENTS/SSD/CN LAB/EXP10]
└─$ gcc -o arp ./arp.c

┌──(gap㉿gapware-pc)-[/DOCUMENTS/SSD/CN LAB/EXP10]
└─$ sudo ./arp
Interface index is 2
Successfully got eth0 MAC address::: C8:D9:D2:AC:7E:84
ARP_REQ MAC address::: C8:D9:D2:AC:7E:84
SOCKET_ADDRESS MAC address::: C8:D9:D2:AC:7E:84

SENT ARP_REQ:::
        FF FF FF FF FF FF C8 D9 D2 AC 7E 84 08 06 00 01 08
        00 06 04 00 01 C8 D9 D2 AC 7E 84 C0 A8 01 1A 00
        00 00 00 00 00 C0 A8 01 01

RECEIVED ARP_RESP::: len=60
Sender IP:::    192.168.1.1
Sender MAC:::    18: 0F: 76: C7: 2B: 22
Receiver IP:::   192. 168. 1. 26
Self MAC:::      C8: D9: D2: AC: 7E: 84

┌──(gap㉿gapware-pc)-[/DOCUMENTS/SSD/CN LAB/EXP10]
└─$ 
```

Result:

The required code for the ARP Implementation Using UDP was written in the Local environment and successfully compiled.