EXPERIMENT: 31

IMPLEMENTATION OF ARP PROTOCOLS IN JAVA/C

Aim: To implement ARP protocols in JAVA/C.

Algorithm:

STEP1: Start

STEP 2: Declare the variables and structure for the socket

STEP 3: Specify the family, protocol, IP address and port number STEP

4: Create a socket using socket() function

STEP 5: Call memcpy() and strcpy functions STEP 6:

Display the MAC address

STEP 7: Stop

Implementing the ARP (Address Resolution Protocol) protocol in C involves constructing and sending ARP request and response packets, as well as handling the reception and processing of ARP packets.

Steps involved:

1. Define the necessary structures and constants:

 - Define the structure for the ARP header, including fields like hardware type, protocol type, hardware address length, protocol address length, operation, sender hardware address, sender protocol address, target hardware address, and target protocol address.

 - Define constants for the ARP hardware type (e.g., Ethernet) and protocol type (e.g., IPv4).

2. Create and send an ARP request packet:

 - Create a socket using the `socket()` function with the `AF_PACKET` address family and `SOCK_RAW` socket type.

 - Create an ARP request packet using the defined ARP header structure.

 - Set the appropriate values for the fields in the ARP header, such as the hardware type, protocol type, operation (request), sender hardware and protocol addresses, and target protocol address.

 - Create and set a `struct sockaddr_ll` structure to specify the interface index, destination MAC address, and other necessary information for sending the packet.

 - Use the `sendto()` function to send the ARP request packet using the socket descriptor and the destination address structure.

3. Receive and process ARP packets:

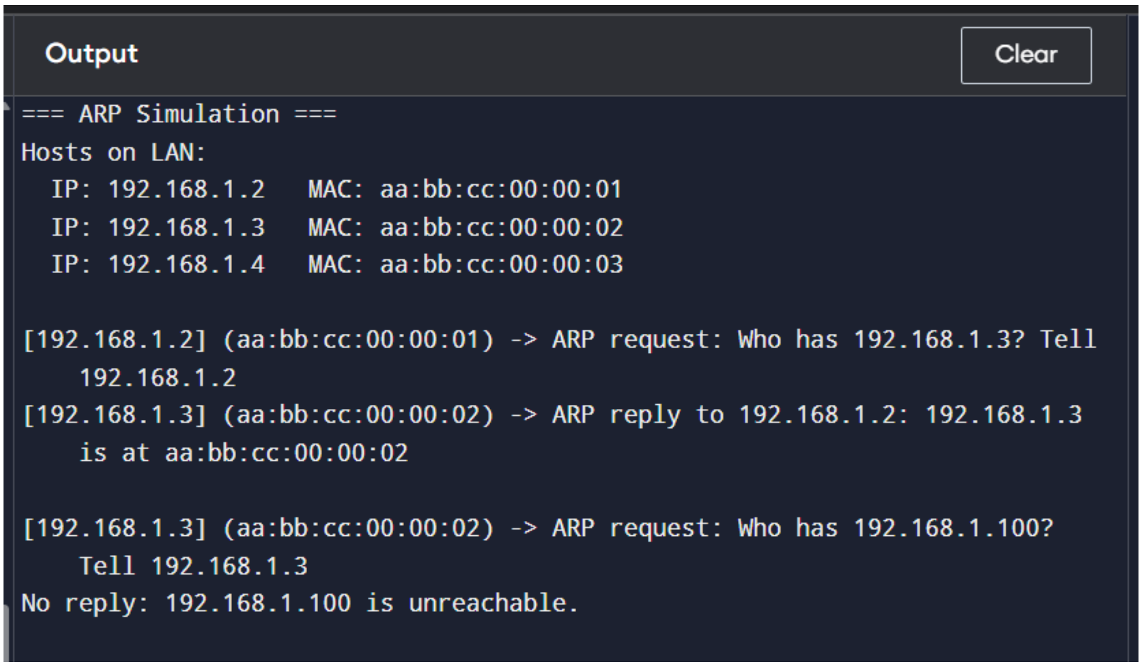
- Create a socket using the `socket()` function with the `AF_PACKET` address family and

`SOCK_RAW` socket type.

- Use a loop to continuously receive packets using the `recvfrom()` function and the socket descriptor.
- Extract the received packet and analyze its contents.
- Parse the ARP header from the received packet and examine the operation field to determine if it is an ARP request or response.
- Process the ARP packet accordingly, such as updating an ARP cache or responding to ARP requests.

Note: Implementing the full functionality of ARP, including caching, table management, and handling ARP requests and responses in a network environment, can be complex. The above steps provide a high-level overview, but a complete implementation would require further details and handling of specific use cases.

Remember to include the necessary header files (`<stdio.h>`, `<stdlib.h>`, `<string.h>`, `<sys/socket.h>`, `<netinet/if_ether.h>`, etc.) and handle errors appropriately in your code. Additionally, consider working with lower-level networking libraries or using a network packet capture library like libpcap for handling raw sockets and packet capturing.



```
=== ARP Simulation ===
Hosts on LAN:
  IP: 192.168.1.2   MAC: aa:bb:cc:00:00:01
  IP: 192.168.1.3   MAC: aa:bb:cc:00:00:02
  IP: 192.168.1.4   MAC: aa:bb:cc:00:00:03

[192.168.1.2] (aa:bb:cc:00:00:01) -> ARP request: Who has 192.168.1.3? Tell
    192.168.1.2
[192.168.1.3] (aa:bb:cc:00:00:02) -> ARP reply to 192.168.1.2: 192.168.1.3
    is at aa:bb:cc:00:00:02

[192.168.1.3] (aa:bb:cc:00:00:02) -> ARP request: Who has 192.168.1.100?
    Tell 192.168.1.3
No reply: 192.168.1.100 is unreachable.
```

Result: Thus the ARP protocols in C are implemented successfully.