Group Number: __________________________

Member Names: __________________________ __________________________

________________________ __________________________

Lab Goals

1. Understand the PF_PACKET protocol family sockets.

2. Understand the difference between the SOCK_RAW packet sockets, and the SOCK_DGRAM packet sockets.

3. Write a raw sockets program to capture all IPV4 packets on a network and print out the IP Header.

4. Write a program to construct an ARP request packet, and wait for the ARP reply.

1 Packet Sockets

The sockets API allows for programs to capture packets before they are processed by the protocol stack. This allows the program to receive the entire packet, including all protocol headers. This is called the packet socket interface. In order to use this type of socket, the process must have super-user or root privilege.

Packet sockets can be configured in two ways. First is the so-called raw socket. For raw sockets, the layer 2 (MAC layer) header is included in the data received by the program. Similarly, when sending a packet on a raw socket, the MAC layer header must be included in the data.

The second way is the cooked mode, which removes the MAC layer header from the packet before being delivered to the application. In cooked mode, there is a socket address structure (discussed later) that allows the program to find out the contents of the MAC layer header.

For either of the packet socket modes, you need to include the following include files to get access to the various types needed.
Using Cooked Mode Packet Sockets

The create a cooked mode packet socket, use the `socket` call, with the following parameters:

```c
int s = socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_ALL))
```

The first parameter (PF_PACKET) indicates we want to use the packet socket interface. The second parameter (SOCK_DGRAM) indicates we want the cooked mode packet socket. The final parameter indicates which layer 3 protocol number we want to receive and send. In the example above the ETH_P_ALL indicates all protocols. We could restrict it to only an individual one, such as ETH_P_IP for IPV4, or ETH_P_ARP for ARP. A complete list of protocols can be found in `linux/if_ether.h` on your Linux system.

When using the cooked–mode packet socket, the `sockaddr` structure to be used is of type `struct sockaddr_ll`. The ll means link layer, so the address indicates a link layer (MAC) address. The definition for this structure is found in `linux/if_packet.h` and is as follows.

```c
struct sockaddr_ll
{
    unsigned short sll_family;
    __be16 sll_protocol;
    int sll_ifindex;
    unsigned short sll_hatype;
    unsigned char sll_pkttype;
    unsigned char sll_halen;
    unsigned char sll_addr[8];
};
```

In the above, the `unsigned short` and `__be16` types are 16 bit quantities, the `int` type is 32 bits, and the `unsigned char` types are 8 bits.

The various fields of this structure are:
1. sll_family is the protocol family. This will always be PF_PACKET when using the packet socket interface.

2. sll_protocol is the protocol found in the layer 2 (MAC) header for this packet. These are the codes found in linux/if_ether.h. This will match the protocol number specified on the third parameter to the socket call, unless that was ETH_P_ANY.

3. sll_ifindex is the interface index where the packet was received (or the index to use when sending a packet). On linux, each hardware interface has a unique interface index.

4. sll_hatype is the hardware address type. This the same value used by the ARP header in the hardware type. The value of 1 indicates an Ethernet.

5. sll_pkttype is the type of packet received. These are enumerated in linux/if_packet.h. We can tell if the packet was addressed to this host, was a broadcast, multicast, or loopback.

6. sll_halen is the hardware address length. For an Ethernet this is 6 since the MAC address for Ethernet is 6 bytes (48 bits).

7. sll_addr is the actual MAC address (source) from the MAC layer header. This is defined as 8 bytes in the structure, but only sll_halen bytes are used.

To receive packets on a cooked–mode packet socket, use the recvfrom socket call as follows:

```c
struct sockaddr_ll linkLayerAddr;
memset(&linkLayerAddr, 0, sizeof(linkLayerAddr));
int sockaddr_len = sizeof(linkLayerAddr);
int actual = recvfrom(packet_socket, buf, sizeof(buf), 0, (struct sockaddr*)&linkLayerAddr, &sockaddr_len);
printf("Got pkt, actual %d\n", actual);
printf("sll_protocol %04x\n", ntohs(linkLayerAddr.sll_protocol));
printf("sll_ifindex %04x\n", ntohs(linkLayerAddr.sll_ifindex));
printf("sll_hatype %04x\n", linkLayerAddr.sll_hatype);
printf("sll_pkttype %04x\n", linkLayerAddr.sll_pkttype);
printf("sll_halen %04x\n", linkLayerAddr.sll_halen);
```

2.1 Lab Assignment.

Write a program using the cooked–mode packet sockets interface that receives packets using the IPV4 protocol. The program should loop infinitely, receiving packets and printing out contents. The program should print the contents of each item in the link layer header, including the source MAC address. Then it should print the first 20 bytes of the packet (the IP header), laid out as four bytes per line (mimicking the way protocol headers are generally depicted. An example output is shown below:
Remember that you need to be the super user in order to run programs using the packet socket interface. You do this by using the su command and entering your root password. Run your program for several minutes, observing the types of packets received. Include a screen shot of your output in the lab turnin documents. Also include a printout of your program.

3 Using Raw Packet Sockets

The create a raw mode packet socket, use the socket call, with the following parameters:

```
int s = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))
```

The parameters are identical to those used for cooked mode sockets, excepting the second parameter is SOCK_RAW indicating a raw mode socket.

Since the MAC layer header is NOT processed prior to delivering the packet, you do not use the struct sockaddr_ll as before. Instead, just use the recv socket call, which will deliver the entire packet, including the MAC layer header.

```
int actual = recv(s, buf, sizeof(buf), 0)
printf("Got pkt, actual %d\n", actual);
```

3.1 Lab Assignment.

Write a program using the raw mode packet sockets interface that receives packets using the IPV4 protocol. The program should loop infinitely, receiving packets and printing out contents. The
program should print the contents of Ethernet header (14 bytes) followed by the 20 byte IP header. Include a screen shot of your output with your turnin documentation. Also include a printout of your program. An example output is shown below:

Got pkt, actual 66
Ethernet Header
00 11 11 c2
46 69 00 0d
66 5d d4 00
08 00
IP Header
45 10 00 34
b8 37 40 00
3e 06 f2 82
8f d7 9d 7b
82 cf e1 d7

4 Sending an ARP Request

In this part of the lab, we will use the cooked packet socket interface to send an ARP request, wait for the reply, and print out the results.

From the class lectures, we know the format of an ARP request and reply packets have a common format, as shown in the figure below. For this part of the lab, we will create a C structure representing the layout of the ARP headers, send an ARP request packet, wait for the reply, and print out the results.

4.1 Lab Assignment.

Write a program that opens a cooked mode packet socket. Clearly, you should start with your program from the previous section, although in this case we want the ARP protocol, not IPV4. Then add a C structure definition representing the ARP request header. It would look something like:

typedef struct {
    __u16 hardwareType; /* 16 bits of hardware type */
    __u16 protocolType; /* Type of protocol this request is for*/
    /* Obviously a lot more here */
} ARP_RequestHeader;
Figure 1: ARP Request Header

In this structure, use `__u8` when you need an 8–bit quantity, `__u16` when you need a 16–bit quantity, and `__u32` when you need a 32–bit quantity. In these types, there are two underscores leading the type name (it looks like one in the pdf file). Use an array of `__u8` values for larger fields such as the 6 byte MAC address. Unfortunately, the C/C++ compiler will always put extra bytes in a structure in order to insure that 32–bit variables are always on a 4–byte boundary. You can see in the layout of the ARP request that the Requester IP Address (which is a 32–bit value), is not aligned on a 4–byte boundary. The way to solve this is to make all fields starting with the Requester MAC Address an array of bytes, such as:

```c
__u8 requesterMacAddress[6];
__u8 requesterIPAddress[4];
__u8 responderMacAddress[6];
__u8 responderIPAddress[4];
```

Then you can just fill in the appropriate values one byte at a time.

To continue with your program, write the code to fill in appropriate values in the ARP request header, and then broadcast the request using the `sendto` socket call. For the responder IP address, you should obviously choose an address for another system on your subnetwork. Your code would look something like this:

```c
struct sockaddr_ll linkLayerAddr;
memset(&linkLayerAddr, 0, sizeof(linkLayerAddr));
/* Fill in the needed values in the linkLayerAddr. */
```
/* At a minimum, you need the protocol, hardware type, packet type */
/* hardware address length, and destination MAC address (which should */
/* be a broadcast). Also note that the sll_ifindex value should be */
/* 1, not zero */

ARP_RequestHeader arpHeader;
/* Fill in ALL the values in the ARP request header here */

int actual = sendto(packet_socket, (char*)&arpHeader, sizeof(arpHeader), 0,
(struct sockaddr*)&linkLayerAddr, sizeof(linkLayerAddr));
printf("Sent pkt, actual %d\n", actual);

Finally, wait for the ARP reply, using recvfrom in a manner identical to the first program. Then
print out the MAC address of the target system returned in the ARP reply.

Include screen shots of your output and a copy of your program in the turnin documentation.

**Summary:** This lab introduces the concept of packet sockets, and gives some practice in reading
and writing packets using this interface.