Universidad de León
Bachelor Degree on Computer Science and Engineering
Course on Computer Networks

CN Practical on PF_PACKET Socket Programming

Reference

Intro to PF Raw sockets:
http://paloalto.unileon.es/cn/labs/PF_PACKETsockets.pdf

Source Code:
http://paloalto.unileon.es/cn/labs/dgramPFPACKETSendv2.c
http://paloalto.unileon.es/cn/labs/rawRec.c

Receiving on a PF_PACKET socket

Receiving on a socket is accomplished by way of the **recvfrom()** system call. That system call can be considered the counterpart to the **sendto()** system call that we studied in a previous practice.

The base source code to the receive program of this practical creates a PF_PACKET/SOCK_RAW socket. The program waits for a block of bytes to arrive and then prints them out assuming that they are all printable. This program creates a socket different than the socket created by the program in the past practical (PF_PACKET/SOCK_DGRAM), thus. In this case, the program, after calling recvfrom(), not only is given access to the received frame's payload, but it is also given access to the frame header. Actually, the char pointer returned by recvfrom() points to the first byte in the header. By iterating over it, we can visit the bytes that make up the header, and the concatenated payload. Do the following practical exercises that will let you better understand the structure of Datalink Frames, their processing in the receiving host and the general notion of multiplexing and encapsulation.

Exercise 1. Assume that the host you're working in is H_1. Check the receive program in host H_2, a host other than the one you're working on now in Lab B6.

- [In host H_2] Login into some host in Lab B6. Copy its main network adaptor's MAC address which you'll paste in a step below.

- [In host H_2] Download the source code of a receive program that uses Raw Sockets (Use the link to it given above), compile it and run it in super-user mode. Let it wait for the send program to send it some data in the next step.

- [In host H_1] Run the send program that we created the past week and have it send some data to the MAC address used by the main adaptor of H_2, which you copied earlier. The data to send you provide it in form of a command-line string in the second argument.

- Check sends and receives with appropriate tcpdumps, akin to those that we used the in the past practical.

- Check that the receive program has received the data sent from H_1.

- Check that all of the lengths printed out by the program make sense.

Exercise 2. Modify the receive program so that it prints out the destination MAC address included in the received frame.

Exercise 3. Modify the receive program so that it prints out the source MAC address included in the received frame.

Exercise 4. Modify the receive program so that it prints out the ethertype field of the received frame. Make sure that you call ntohs() upon the received ethertype field value so that the byte ordering is consistent with that of your host's architecture. The ntohs() library call transforms the byte ordering of an integer that was received over the network (Known as *Network Byte Order, or Big Endian)* to the byte ordering used by the receiving host, which may or may not be the same, depending on the processor architecture. Maybe you want to read ntohs() man page. Finally, make sure that the print out of the ethertype is human-readable.

Exercise 5. What is the purpose of the library call **htons()**? When should you use it?

Exercise 6. Run the receive program and let it run for a while by making sure that no other program will send it any data, thereby guaranteeing that it runs without stop for a while. You can, for example have your receive program **not accept any data sent to the broadcast** address and use a particular ethertype that won't be used by anybody else in the Lab.

- While the program is running, request a new terminal and execute the following command which probes the kernel for sockets that are active at the time:

```
$ ss -packet -epn
```

- Interpret the printout resulting from the execution of the ss command

Exercise 7 [Home]. Extend the **send** program so that it computes the even parity of 1's of the **sent** data (Only 1 parity bit for the whole structure). The parity bit will be encapsulated into one byte which will be concatenated to the data bytes.

Exercise 8 [Home]. Extend the **receive** program so that it computes the even parity of 1's to the received data bytes (Only 1 parity bit for the whole structure). The resulting bit will be compared to the parity bit encapsulated into the last byte received, which was computed and sent by the sender. Then, the program should print out the values of the sent parity bit and the computed parity bit.

Source code to the PF_PACKET/RAW_SOCK receive program

```
/*********************************************************
 * From textbook Conceptual Computer Networks           *
 * All rights reserved (C) 2012-2025 by:                *
 * José María Foces Morán & José María Foces Vivancos   *
 *                                                      *
*********************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include <fcntl.h>
#include <memory.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>

#include <linux/if_ether.h>
#include <linux/if_packet.h>
#include <net/ethernet.h>
#include <net/if.h>

#include <signal.h>
#include <errno.h>
#include <sys/time.h>

#define byte u_char
#define TRUE 1
#define ETHERTYPE_EXPERIMENTAL 0x07ff
#define DEFAULT_MTU 1500


void printDestMac(char *buf){

    printf("Destination MAC:\n");
```

```c
}

void printSourceMac(char *buf){

    printf("Destination MAC:\n");

}

void printEthertype(char *buf){

    printf("Ethertype:\n");

}

void printProgramLegend(char *payload) {

    printf("Receive a frame with PF_PACKET/SOCK_DGRAM\n");
    printf("\tEthertype = %hx\n", ETHERTYPE_EXPERIMENTAL);

    fflush(stdout);

}

/*
 * This function fills the fields of the socket address structure
 * Some of the come from the command line arguments passed by the user
 *
 * u_int16_t is  used for representing the ethertype
 * u_int16_t is declared int /usr/include/x86_64-linux-gnu/sys/types.h
 * with #include <sys/types.h>
 *
 * __be16 is defined in /usr/include/linux/types.h and is also used for
 * representing ethernet's ethertype field
 */
struct sockaddr_ll fillSocketAddress(char *ifName, u_int16_t ethertype) {

    /*
     * sockaddr_ll has a slightly different use when sending than is used for
     * receiving.
     * When sending, sockaddr_ll stores the Destination MAC and the
     * multiplexing key (Ethertype) and the index of the interface to be used for
     * actually transmitting the frame
     *
     * When receiving, sockaddr_ll stores the Source MAC address, the received
     * Ethertype and the interface index the frame was received onto
     */
    struct sockaddr_ll socketAddress;

    socketAddress.sll_family = PF_PACKET;

    /* Index of network interface */

    printf("Device = %s\n\n", ifName);

    socketAddress.sll_ifindex = if_nametoindex(ifName);
    if (socketAddress.sll_ifindex == 0) {
        perror("Error indexing interface name. Exiting.\n");
        exit(-2);
    }

    /* Address length*/
    socketAddress.sll_halen = ETH_ALEN;

    //Ethertype translated to Network Byte Order
    socketAddress.sll_protocol = htons(ethertype);

    //arp-related
    socketAddress.sll_hatype = 0;

    //arp-related
    socketAddress.sll_pkttype = 0;

    return socketAddress;

}
```

```
void start(char *ifName) {

    struct sockaddr_ll socketAddress = fillSocketAddress(ifName, (u_short)
ETHERTYPE_EXPERIMENTAL);

    int sock = socket(PF_PACKET, SOCK_RAW, 0);

    if (bind(sock, (struct sockaddr *) &socketAddress, sizeof(socketAddress)) == -1)
{
            perror("Error on bind().");
            exit(-1);
    }

    printf("Waiting for data to be received onto PF_PACKET socket\n");

    u_char *buf = (u_char *) malloc(DEFAULT_MTU);

    struct sockaddr_ll src_addr;
    socklen_t addrlen;

    ssize_t n;

    if ( (n = recvfrom(sock, buf, DEFAULT_MTU, 0, (struct sockaddr *) &src_addr,
&addrlen)) == -1) {
        printf("\nrecvfrom() call failed\n. Exiting\n");
        exit(-1);
    }

    printf("%d bytes received onto raw socket.\n", n);

    const int etherHeaderLength = 14;
    int i;

    for(i = etherHeaderLength ; i < n; i++){
      printf("%c", buf[i]);
    }

    printf("\ni = %d\n", i);

    printf("Received: ETHER HEADER(%d)|PAYLOAD(%d)\n", etherHeaderLength, n -
etherHeaderLength);

    printDestMac(buf);
    printSourceMac(buf);
    printEthertype(buf);

    fflush(stdout);

}

int main(int argc, char** argv) {


    /*
     * Command-line processing
     */
    if (argc == 2) {

        start(argv[1]); //interface label

    } else {

        fprintf(stderr, "Usage: %s \t<Network Interface>\n", argv[0]);
        exit(-1);

    }

}
```