```
/*****************************************************************************
 practica2.c
 Muestra las direciones Ethernet de la traza que se pasa como primer parametro.
 Debe complatarse con mas campos de niveles 2, 3, y 4 tal como se pida en el enunciado.
 Debe tener capacidad de dejar de analizar paquetes de acuerdo a un filtro.

 Compila: gcc -Wall -o practica2 practica2.c -lpcap, make
 Autor: Marta GarcÃa MarÃn y LucÃa Colmenarejo PÃ©rez
 2017 EPS-UAM
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include <pcap.h>
#include <string.h>
#include <netinet/in.h>
#include <linux/udp.h>
#include <linux/tcp.h>
#include <signal.h>
#include <time.h>
#include <getopt.h>
#include <inttypes.h>

/*Definicion de constantes ************************************************/
#define ETH_ALEN      6      /* Tamanio de la direccion ethernet           */
#define ETH_HLEN      14     /* Tamanio de la cabecera ethernet            */
#define ETH_TLEN      2      /* Tamanio del campo tipo ethernet            */
#define TIPO_IP         2048  /* Corresponde al tipo ip, para detectar IPv4 */
/*El campo de CRC, es un cÃ³digo de detecciÃ³n de errores que ocupa 4 bits   */
#define ETH_FRAME_MAX 1514   /* Tamanio maximo la trama ethernet (sin CRC) */
#define ETH_FRAME_MIN 60     /* Tamanio minimo la trama ethernet (sin CRC) */
#define ETH_DATA_MAX  (ETH_FRAME_MAX - ETH_HLEN) /* Tamano maximo y minimo de los datos de
una trama ethernet*/
#define ETH_DATA_MIN  (ETH_FRAME_MIN - ETH_HLEN)
#define IP_ALEN 4     /* Tamanio de la direccion IP, se referirÃ¡ a 4 bytes= 32 bits*/
#define OK 0
#define ERROR 1
#define PACK_READ 1
#define PACK_ERR -1
#define TRACE_END -2
#define NO_FILTER 0

void analizar_paquete(const struct pcap_pkthdr *hdr, const uint8_t *pack);
void handleSignal(int nsignal);

pcap_t *descr = NULL;
uint64_t contador = 0;
uint8_t ipsrc_filter[IP_ALEN] = {NO_FILTER};
uint8_t ipdst_filter[IP_ALEN] = {NO_FILTER};
uint16_t sport_filter= NO_FILTER;
uint16_t dport_filter = NO_FILTER;

void handleSignal(int nsignal){
       (void) nsignal; // indicamos al compilador que no nos importa que nsignal no se
utilice
       printf("Control C pulsado (%"PRIu64" paquetes leidos)\n", contador);
       pcap_close(descr);
       exit(OK);
}

int main(int argc, char **argv){
       uint8_t *pack = NULL;
       struct pcap_pkthdr *hdr;

       char errbuf[PCAP_ERRBUF_SIZE];
       char entrada[256];
       int long_index = 0, retorno = 0;
       char opt;
```

```
	(void) errbuf; //indicamos al compilador que no nos importa que errbuf no se
utilice. Esta linea debe ser eliminada en la entrega final.

	if (signal(SIGINT, handleSignal) == SIG_ERR) {
		printf("Error: Fallo al capturar la senal SIGINT.\n");
		exit(ERROR);
	}
	if (argc > 1) {
		if (strlen(argv[1]) < 256) {
			strcpy(entrada, argv[1]);
		}
	} else {
		printf("Ejecucion: %s <-f traza.pcap / -i eth0> [-ipo IPO] [-ipd IPD] [-po
PO] [-pd PD]\n", argv[0]);
		exit(ERROR);
	}
	static struct option options[] = {
		{"f", required_argument, 0, 'f'},
		{"i",required_argument, 0,'i'},
		{"ipo", required_argument, 0, '1'},
		{"ipd", required_argument, 0, '2'},
		{"po", required_argument, 0, '3'},
		{"pd", required_argument, 0, '4'},
		{"h", no_argument, 0, '5'},
		{0, 0, 0, 0}
	};

	//Simple lectura por parametros por completar casos de error, ojo no cumple 100% los
requisitos del enunciado!
	while ((opt = getopt_long_only(argc, argv, "f:i:1:2:3:4:5", options, &long_index))
!= -1) {
		switch (opt) {
		case 'i' :
			if(descr) { // comprobamos que no se ha abierto ninguna otra interfaz
o fichero
				printf("Ha seleccionado mÃ¡s de una fuente de datos\n");
				pcap_close(descr);
				exit(ERROR);
			}

			if ((descr = pcap_open_live(optarg, 1514, 0, 10, errbuf)) == NULL){
				printf("Error: pcap_open_online() Interface: %s, %s %s %d.\n",
optarg,errbuf,__FILE__,__LINE__);
				exit(ERROR);
			}
			break;

		case 'f' :
			if(descr) { // comprobamos que no se ha abierto ninguna otra interfaz
o fichero
				printf("Ha seleccionado mÃ¡s de una fuente de datos\n");
				pcap_close(descr);
				exit(ERROR);
			}

			if ((descr = pcap_open_offline(optarg, errbuf)) == NULL) {
				printf("Error: pcap_open_offline(): File: %s, %s %s %d.\n",
optarg, errbuf, __FILE__, __LINE__);
				exit(ERROR);
			}

			break;

		case '1' :
			if (sscanf(optarg, "%"SCNu8".%"SCNu8".%"SCNu8".%"SCNu8"",
&(ipsrc_filter[0]), &(ipsrc_filter[1]), &(ipsrc_filter[2]), &(ipsrc_filter[3])) != IP_ALEN)
{
				printf("Error ipo_filtro. Ejecucion: %s /ruta/captura_pcap [-
ipo IPO] [-ipd IPD] [-po PO] [-pd PD]: %d\n", argv[0], argc);
				exit(ERROR);
			}
```

```
                        break;

                case '2' :
                        if (sscanf(optarg, "%"SCNu8".%"SCNu8".%"SCNu8".%"SCNu8"",
&(ipdst_filter[0]), &(ipdst_filter[1]), &(ipdst_filter[2]), &(ipdst_filter[3])) != IP_ALEN)
{
                                printf("Error ipd_filtro. Ejecucion: %s /ruta/captura_pcap [-
ipo IPO] [-ipd IPD] [-po PO] [-pd PD]: %d\n", argv[0], argc);
                                exit(ERROR);
                        }

                        break;

                case '3' :
                        if ((sport_filter= atoi(optarg)) == 0) {
                                printf("Error po_filtro.Ejecucion: %s /ruta/captura_pcap [-ipo
IPO] [-ipd IPD] [-po PO] [-pd PD]: %d\n", argv[0], argc);
                                exit(ERROR);
                        }

                        break;

                case '4' :
                        if ((dport_filter = atoi(optarg)) == 0) {
                                printf("Error pd_filtro. Ejecucion: %s /ruta/captura_pcap [-
ipo IPO] [-ipd IPD] [-po PO] [-pd PD]: %d\n", argv[0], argc);
                                exit(ERROR);
                        }

                        break;

                case '5' :
                        printf("Ayuda. Ejecucion: %s <-f traza.pcap / -i eth0> [-ipo IPO] [-
ipd IPD] [-po PO] [-pd PD]: %d\n", argv[0], argc);
                        exit(ERROR);
                        break;

                case '?' :
                default:
                        printf("Error. Ejecucion: %s <-f traza.pcap / -i eth0> [-ipo IPO] [-
ipd IPD] [-po PO] [-pd PD]: %d\n", argv[0], argc);
                        exit(ERROR);
                        break;
                }
        }

        if (!descr) {
                printf("No selecciono ningÃ°n origen de paquetes.\n");
                return ERROR;
        }

        //Simple comprobacion de la correcion de la lectura de parametros
        printf("Filtro:");
        //if(ipsrc_filter[0]!=0)
        printf("ipsrc_filter:%"PRIu8".%"PRIu8".%"PRIu8".%"PRIu8"\t", ipsrc_filter[0],
ipsrc_filter[1], ipsrc_filter[2], ipsrc_filter[3]);
        //if(ipdst_filter[0]!=0)
        printf("ipdst_filter:%"PRIu8".%"PRIu8".%"PRIu8".%"PRIu8"\t", ipdst_filter[0],
ipdst_filter[1], ipdst_filter[2], ipdst_filter[3]);

        if (sport_filter!= NO_FILTER) {
                printf("po_filtro=%"PRIu16"\t", sport_filter);
        }

        if (dport_filter != NO_FILTER) {
                printf("pd_filtro=%"PRIu16"\t", dport_filter);
        }

        printf("\n\n");
```

```
	do {
		retorno = pcap_next_ex(descr, &hdr, (const u_char **)&pack);

		if (retorno == PACK_READ) { //Todo correcto
			contador++;
			/*ImpresiÃ³n del nÃºmero del paquete, no es necesariam, pero sÃ
aconsejable de cara al examen*/
			printf("..................................................\n");
			printf("Paquete nÃºmero %"PRIu64"\n", contador);
			printf("..................................................\n");
			analizar_paquete(hdr, pack);

		} else if (retorno == PACK_ERR) { //En caso de error
			printf("Error al capturar un paquetes %s, %s %d.\n",
pcap_geterr(descr), __FILE__, __LINE__);
			pcap_close(descr);
			exit(ERROR);

		}
	} while (retorno != TRACE_END);

	printf("..................................................\n");
	printf("\nSe procesaron %"PRIu64" paquetes.\n\n", contador);
	pcap_close(descr);
	return OK;
}


void analizar_paquete(const struct pcap_pkthdr *hdr, const uint8_t *pack){
	uint16_t tipoEthernet=0;	/*lo declaramos para guardar el tipo de protocolo*/
	uint8_t version=0;		/*lo declaramos para guardar la versiÃ³n*/
	uint8_t ihl=0;			/*lo declaramos para guardar el ihl*/
	uint16_t longitud = 0;		/*lo declaramos para guardar la longitud total*/
	uint8_t protocolo = 0;		/*lo declaramos para guardar el protocolo*/
	uint16_t posicion = 0;		/*lo declaramos para guardar la
posicion/desplazamiento*/
	uint8_t ack = 0;		/*lo declaramos para guardar la flag de ACK*/
	uint8_t syn = 0;		/*lo declaramos para guardar la flag de SYN*/
	uint16_t puerto_origen = 0;	/*lo declaramos para guardar el puerto de origen*/
	uint16_t puerto_destino = 0;	/*lo declaramos para guardar el puerto de destino*/
	uint16_t long_udp = 0;		/*lo declaramos para guardar la longitud del UPD*/


	printf("\nNuevo paquete capturado el %s\n", ctime((const time_t *) & (hdr-
>ts.tv_sec)));

	/**********************Imprimimos los campos de la capa 2**********************/

	printf("---Capa 2---\n");

	int i = 0;
	printf("Direccion ETH destino= ");
	printf("%02X", pack[0]);

	for (i = 1; i < ETH_ALEN; i++) {
		printf(":%02X", pack[i]);
	}

	printf("\n");
	pack += ETH_ALEN;

	printf("Direccion ETH origen = ");
	printf("%02X", pack[0]);

	for (i = 1; i < ETH_ALEN; i++) {
		printf(":%02X", pack[i]);
	}

	printf("\n");
	pack += ETH_ALEN;
```

```
	printf("Tipo de protocolo: ");
	printf("0x%02X%02X\n", pack[0], pack[1]);
	printf("\n");

	tipoEthernet = ntohs(*(uint16_t*)pack); /*Lo que intento hacer aqui es leer el valor
de los dos bytes(16 bits) y ver si el valor esperado se corresponde con 2048 que es el valor
de 0x0800 en hexadecimal*/

	pack += ETH_TLEN;

	/*********************Imprimimos los campos de la capa 3*********************/

	printf("---Capa 3---\n");

	if(tipoEthernet == TIPO_IP){
		printf("Se trata del protocolo IPv4.\n");

		version = *pack>>4;
		printf("Version: %u \n",version);
		ihl = (*pack & 0x0F)<<2;
		printf("IHL: %u \n", ihl);

		pack += 2;
		longitud = ntohs(*(uint16_t*)pack);
		printf("Longitud total: %u\n", longitud);

		pack += 4;
		posicion= (ntohs(*(uint16_t*)pack)&0x1FFF)<<3;
		printf("Posicion: %u\n", posicion);

		pack += 2;
		printf("Tiempo de vida: %u\n", pack[0]);

		pack += 1;
		protocolo = pack[0];
		printf("Protocolo/desplazamiento: %u\n", protocolo);


		if(protocolo != 6 && protocolo != 17){
			printf("No es el protocolo esperado\n");
			return;
		}

		pack += 3;

		/*Control para tener en cuenta la posible direcciÃ³n(ip) de origen que nos
pueden pasar de parÃ¡metro*/
		if (ipsrc_filter[0] != 0 && ipsrc_filter[1] != 0 && ipsrc_filter[2] != 0 &&
ipsrc_filter[3] != 0){
			if (pack[0] != ipsrc_filter[0] || pack[1] != ipsrc_filter[1] ||
pack[2] != ipsrc_filter[2] || pack[3] != ipsrc_filter[3]){
				printf("\nÂ¡Â¡No se corresponde con la direccion origen del
filtro!!\n");
				printf("\n");
				return;
			}
		}
		printf("Direccion origen: %u.%u.%u.%u\n", pack[0],pack[1],pack[2],pack[3]);


		pack += 4;

		/*Control para tener en cuenta la posible direcciÃ³n(ip) de destino que nos
pueden pasar de parÃ¡metro*/
		if (ipdst_filter[0] != 0 && ipdst_filter[1] != 0 && ipdst_filter[2] != 0 &&
ipdst_filter[3] != 0){
			if (pack[0] != ipdst_filter[0] || pack[1] != ipdst_filter[1] ||
pack[2] != ipdst_filter[2] || pack[3] != ipdst_filter[3]){
				printf("\nÂ¡Â¡No se corresponde con la direccion destino del
filtro!!\n");
				printf("\n");
```

```
                        return;
                }
        }
        printf("Direccion destino:%u.%u.%u.%u\n", pack[0],pack[1],pack[2],pack[3]);

        if(posicion != 0x0000){
                printf("El paquete IP no tiene capa de transporte(nivel 4)\n\n");
                return;
        }

        pack += 4;

        if(ihl != 20){
                pack += (ihl - 20);
        }

/*********************Imprimimos los campos de la capa 4*********************/

        printf("\n---Capa 4---\n");
        puerto_origen = htons(*(uint16_t*)pack);

        /*Control para tener en cuenta el posible puerto de origen que nos pueden
pasar de parÃ¡metro*/
        if (sport_filter != 0){
                if (sport_filter != puerto_origen){
                        printf("\nÂ¡Â¡No se corresponde con el puerto de origen del
filtro!!\n");
                        printf(" \n");
                        return;
                }
        }

        printf("Puerto de origen: %u\n", puerto_origen );
        pack += 2;


        puerto_destino =htons(*(uint16_t*)pack);
        /*Control para tener en cuenta el posible puerto de destino que nos pueden
pasar de parÃ¡metro*/
        if (dport_filter != 0){
                if (dport_filter != puerto_destino){
                        printf("\nÂ¡Â¡No se corresponde con el puerto de destino del
filtro!!\n");
                        printf("\n");
                        return;
                }
        }
        printf("Puerto de destino: %u\n", puerto_destino );
        pack += 2;

        if(protocolo == 6){
                pack+=9;
                ack = (*pack & 0x10) >> 4;
                printf("ACK: %u\n", ack);
                syn = (*pack & 0x02) >> 1;
                printf("SYN: %u\n", syn);
        }
        if(protocolo == 17){
                long_udp =htons(*(uint16_t*)pack);
                printf("Longitud UDP: %u\n\n", long_udp);
        }


}else{
        printf("No es el protocolo esperado\n");
        return;
}

}
```