Entorno de programación de nivel 1: La librería PCAP

La librería libpcap nos permite capturar paquetes desde un programa C. En sistemas Windows, la librería se llama Winpcap.

Para compilar cualquier programa que haga uso de la librería libpcap necesitamos incluir el fichero de cabecera pcap.h en las fuentes (#include <pcap.h>) y añadir en la cadena de compilación la orden –lpcap.

También se puede usar el makefile junto con el resto de archivos de cabecera y librería que se ha colgado en el Moodle de la asignatura.

Para instalarla la librería en un sistema basado en Debian (Ubuntu por ejemplo) se puede ejecutar: sudo apt_get install libpcap_dev

Veamos primero las funciones básicas que nos permite la libpcap:

Capturar de un interfaz

Para abrir un interfaz para captura:

pcap_t *pcap_open_live(const char *device, int snaplen,int promisc, int to_ms, char *errbuf);

- Device: es el nombre de la interfaz que se quiere abrir (eth0, eth1...).
- snaplen: es la cantidad de bytes que se quieren guardar por cada paquete. Es útil cuando no nos interesa la carga útil del paquete y así reduciríamos el tamaño de la captura. Por defecto usaremos el valor 1514 (MTU Ethernet).
- promisc: indica si queremos abrirla en modo promiscuo (promisc=1) o no (promisc=0).
- to_ms duración del timeout de lectura. Tiempo que se espera para leer varios paquetes en una misma transacción.
- En errbuf se guarda el mensaje de error, si procede.

La función nos devuelve el puntero al descriptor de fichero pcap. Este descriptor es necesario cuando vayamos a realizar alguna operación sobre el tráfico. Por ejemplo leer un paquete. A efectos prácticos esta estructura es similar a un FILE *

Hay que recordar que para abrir una interfaz es necesario tener permisos de superusuario.

Ejemplo:

p=pcap_open_live(“eth0”,0,1, 100, errbuf);

Abre para la interfaz eth0 en modo promiscuo, capturando el paquete en su totalidad, con un timeout de lectura de 100 ms.

En caso de error, guarda el mensaje en la cadena errbuf. En esta práctica no se realizarán capturas sobre interfaces.

Recibir tramas:

Para la recepción de tramas vamos a utilizar la función:

int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user);

- p: es un puntero a una estructura pcap_t previamente rellena por la función pcap_open_live.
- callback: función que será ejecutada cada vez se reciba una trama. Esta función es la función de notificación de nivel 1. Esta función tiene la siguiente interfaz:
    - typedef void (*pcap_handler)(u_char *user, const struct pcap_pkthdr *h, const u_char *bytes);
- user: información a nivel de usuario que se desea pasar cada vez que se reciba una trama. Si no se desea pasar nada se debe rellenar con NULL.

Ejemplo:

```
void attendPacket(u_char *user,const struct pcap_pkthdr *h,const u_char *packet)
{
        printf("Paquete recibido\n");
}
Int main(int argc,char *argv[])
{
if((p=pcap_open_live("eth0",1514,1,timeout, errbuff))==NULL)
        return -1;
pcap_loop(p,-1,attendPacket,NULL);
pcap_close(p);
}
```

ATENCIÓN: La función pcap_loop es bloqueante, es decir, una vez llamemos a la función no se ejecutará el código que está a continuación antes que no salgamos del loop de captura. Para salir del loop de captura se debe utilizar la función pcap_breakloop.

void pcap_breakloop(pcap_t *p);

- p: descriptor pcap que se está utilizando para la captura.

La función pcap_breakloop debe ser llamada en la función FinalizarNivel1a.

Enviar paquetes:

Para el envío de tramas Ethernet vamos a usar la función:

int pcap_sendpacket(pcap_t *p, const u_char *buf, int size);

- p: es un puntero a una estructura pcap_t previamente rellena por la función pcap_open_live.
- buf: Buffer donde está almacenado el paquete que se desea enviar.
- size: Tamaño del paquete a enviar.

La función devuelve el número de bytes que han sido correctamente enviados.

Cerrar Captura:

Las capturas abiertas con pcap_open_live, se cierran con pcap_close.

void pcap_close(pcap_t *p);

- p es el fichero a cerrar

Uso de hilos:

La función pcap_loop debe ser llamada en IniciarNivel1a. Para evitar el bloqueo en la función de inicialización se propone hacer uso de hilos. Los hilos permiten ejecutar tareas concurrentemente en el S.O. Para lanzar un hilo se utiliza la función:

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)

(void *), void *arg);

- thread: estructura de hilo que será rellena por la función pthread_create. Esta estructura se usará posteriormente para comprobar la finalización del hilo.
- Attr: atributos de ejecución del hilo. En nuestro caso pasaremos NULL para utilizar los atributos por defecto.
- Start_routine: puntero a función que se ejecutará en el nuevo hilo que se cree. Esta función debe tener el siguiente prototipo:
    - void * threadFunction(void *args)
- arg: conjunto de argumentos que serán pasados a la función anterior (start_routine) como primer argumento. En nuestro caso no hará falta usar este valor y se puede pasar a NULL.

Es importante entender el concepto que hay detrás de los hilos. Cuando se llama a la función pthread_create se crean 2 flujos de ejecución en paralelo. El primero continúa con el programa principal ejecutando las instrucciones posteriores a pthread_create. El segundo ejecutará en paralelo la función pasada como segundo argumento. Una vez que dicha función retorna, se considera que el hilo ha terminado. Para comprobar que el hilo ha acabado se usa la función:

int pthread_join(pthread_t t,void **retval);

- t:Identificador de hilo del cual queremos comprobar su finalización.
- retval: valor de retorno de la función pasada como segundo argumento a pthread_create. En nuestro caso este valor será NULL ya que la función de hilo no debe devolver nada.

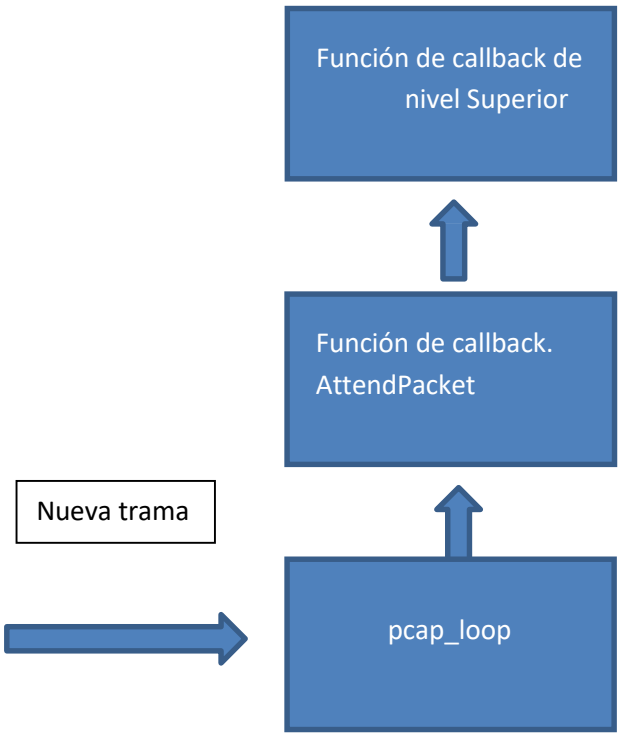
Ejemplo:

```
void * startCapture(void *arg)
{
  printf("Esto es el hilo y se ejecuta en paralelo\n");
}


pthread_t tid;
int main(int argc,char *argv[])
{

        printf("iniciamos el programa y vamos a crear un hilo\n");
        pthread_create(&tid,NULL,startCapture,NULL);
        printf("Esto es el main\n");
        pthread_join(tid,NULL);
        return 0;

}
```

Para utilizar hilos en un programa en C se debe hacer un include de pthread.h y compilar utilizando la opción –lpthread.

¿Cómo funciona la recepción de un paquete en el nivel1a?

A continuación se presenta un gráfico de llamadas que representa las funciones que se llaman cada vez que se recibe un paquete.



NOTA FINAL: La comunicación entre la parte emisora y la parte receptora de un programa debe realizarse a través de variables globales. Estas variables globales deben ser protegidas con semáforos (pthread_mutex) si van a ser accedidas concurrentemente. Hay que tener en cuenta que la parte receptora y la emisora ejecutarán en paralelo generalmente ya que llamaremos a la función pcap_loop dentro de un nuevo hilo.