PRÁCTICA 1

NIVEL DE ENLACE. INTRODUCCIÓN A ETHERNET Y PUESTA A PUNTO DEL SOFTWARE DE TRABAJO.

Prácticas de Fundamentos de Redes y Comunicaciones – Curso 2020/2021

David Cortés Polo (dcorpol@unex.es)
María del Mar Ávila Vegas (mmavila@unex.es)

08 de febrero de 2021

Índice

1. Objetivos.

Las primeras sesiones prácticas de la asignatura, tienen como objetivo la familiarización con el nivel de enlace, así como con la tecnología Ethernet y el formato de sus tramas. Igualmente, se describe el software necesario para trabajar en las sesiones de prácticas, así como la librería que se va a usar para el desarrollo de las mismas.

2. Introducción al modelo OSI.

Es un estándar desarrollado en 1980 por la ISO (Organización internacional para la Estandarización) que define un modelo formado por 7 niveles o capas, mediante el cual se especifican las diferentes fases por las que deben pasar los datos para viajar de un dispositivo a otro sobre una red de comunicaciones. Cada nivel posee su propia función y su propia unidad de datos (figura 1).
El objetivo de este modelo es interconectar sistemas de diferentes fuentes para que puedan intercambiar información sin ningún problema.

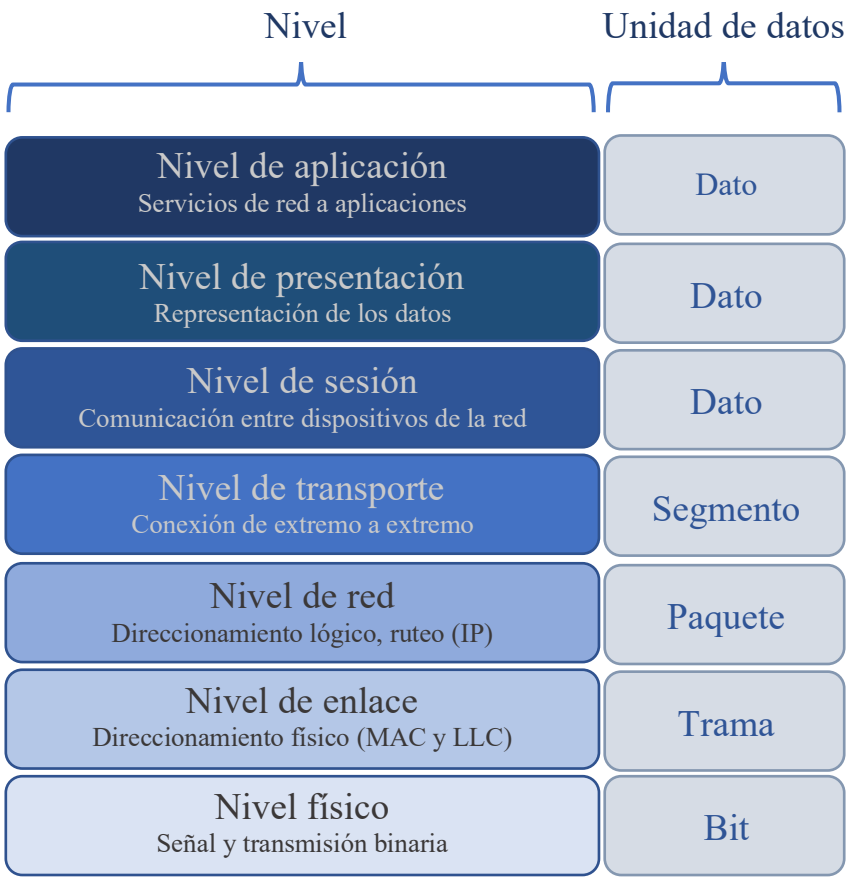

Figura 1: Niveles del modelo OSI.

3. Encapsulación y desencapsulación.

Para transmitir un mensaje de usuario desde un emisor hasta un receptor, el modelo OSI **encapsula** la información desde el nivel de aplicación hasta el nivel físico del emisor. Posteriormente se lleva a cabo el proceso contrario, la **desencapsulación** de los datos, los datos se van desencapsulando desde el nivel físico hasta el nivel de aplicación del receptor, donde finalmente se entregaría el mensaje al usuario (figura 2). El proceso se describe a continuación:

1. Los datos del usuario se envían al nivel de aplicación.
2. El nivel de aplicación añade una cabecera propia a los datos de usuario y éstos se envían al nivel de presentación.

3. El nivel de presentación añade su propia cabecera pasando todos los datos al nivel de sesión.
4. El nivel de sesión añade su cabecera entregando estos datos al nivel de transporte.
5. El nivel de transporte, igualmente agrega su cabecera y sus datos pasan al nivel de red.
6. El nivel de red añade su cabecera y estos datos se entregan al nivel de enlace de datos.
7. El nivel de enlace añade su cabecera y una cola (tráiler). Esta última se utiliza para controlar la integridad de los datos (FCS - Frame Check Secuence). Es empleada para detectar en el receptor si los datos llegaron con errores o no. Este conjunto de datos se pasaría al nivel físico.
8. El nivel físico envía los datos en forma de bits a través del medio de transmisión.
9. A continuación, se realizaría el proceso de desencapsulación. Las secuencias de bits del nivel físico se transfieren al nivel de enlace. Éste verifica la información contenida en la cola (FCS) y si encontrase algún error descartaría los datos y solicitaría su reenvío. Si no hubiera errores, la capa de enlace de datos leería e interpretaría la información de control contenida en la cabecera de nivel de enlace, eliminaría la cabecera y la cola y transferiría los datos al nivel de red.
10. De esta manera, los datos se van entregando a los niveles superiores del receptor, en el que cada nivel se ocupa de extraer la cabecera que anteriormente había añadido su nivel homólogo, la interpretaría y facilitaría la unidad de datos al nivel superior.
11. Finalmente, los datos llegarán a la capa de aplicación, los cuales se entregarán como mensaje al usuario receptor.

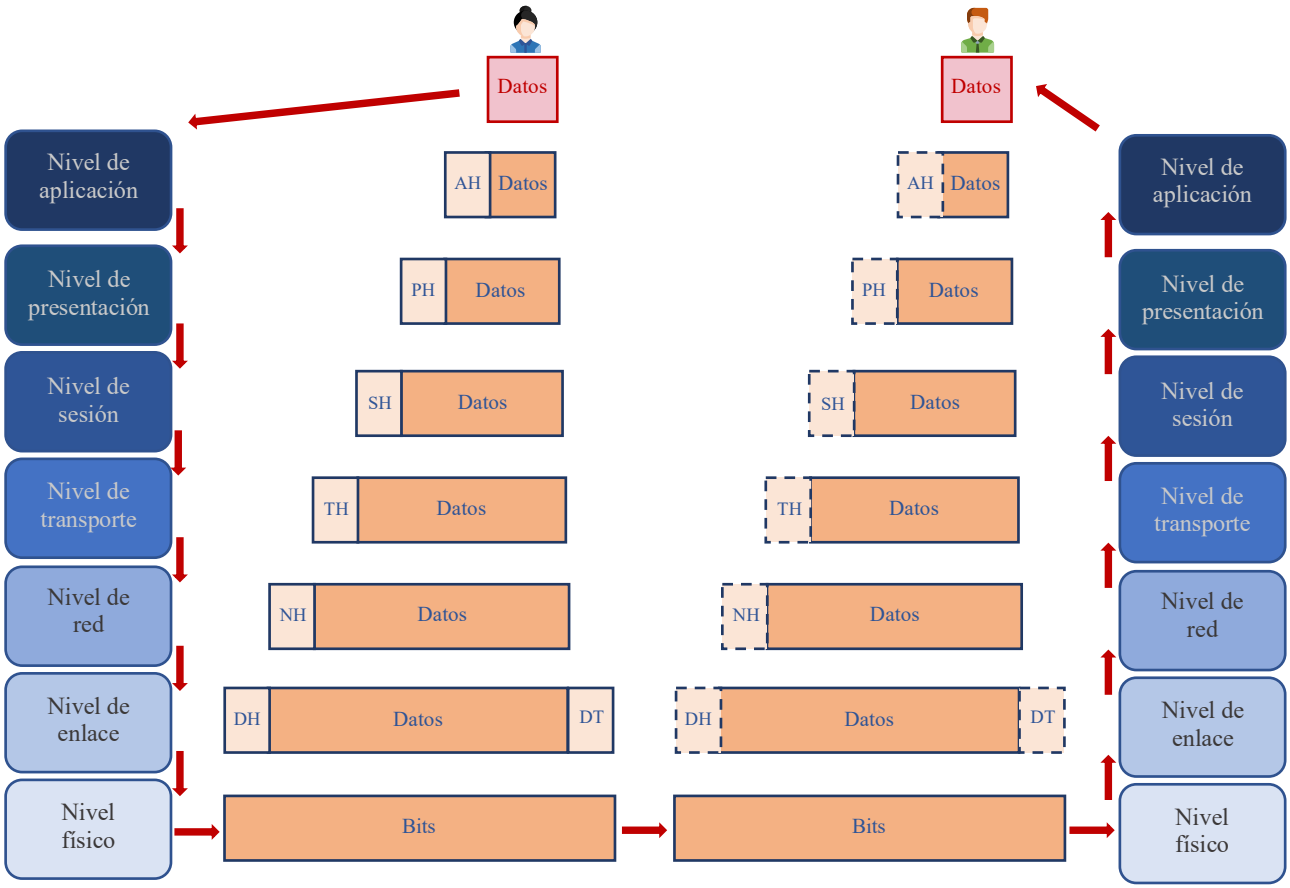

Figura 2: Encapsulación y desencapsulación en el modelo OSI.

4. Nivel de enlace.

El nivel de enlace se encuentra entre el nivel físico y el de red y presta servicios al nivel de red, usando los servicios proporcionados por el nivel físico. Se ocupa de suministrar un transporte fiable de bits a la capa de red, solventando los posibles errores producidos por las deformaciones del canal, ruido de transmisión, y otras perturbaciones causadas en el nivel físico. Este nivel, se encarga de que los bits emitidos por el nivel físico, lleguen a su destino en el mismo orden en que se enviaron; podrían producirse errores o pérdida de bits, pero éstos siempre llegarán en el mismo orden.

Las principales funciones que desarrolla el nivel de enlace son las siguientes:

- Agrupa los bits en grupos denominados tramas. El nivel de enlace de cada nodo se encarga de encapsular el paquete que proviene del nivel de red en una trama antes de enviarlo al siguiente nodo. La construcción de tramas dependerá del formato específico del protocolo de nivel de enlace. El "entramado" permite delimitar el inicio y el fin de cada trama para que el receptor pueda reconocer y procesar cada trama de forma individual.

- Realiza un control de errores. Se encarga de detectar el posible error y analizar si puede ser corregido en el receptor o descartar y retransmitir la trama por el nodo emisor.

- Efectúa control de flujo para evitar la pérdida de información. Si el emisor envía una mayor cantidad de información de la que el receptor puede procesar, se podría comunicar al emisor que bajase el ritmo de envío o que dejase momentáneamente de transmitir.

No todas las funciones se implementan en todos los protocolos de enlace. La retransmisión de tramas erróneas y el control de flujo a menudo se implementan en las superiores (capa de red, de transporte, o incluso en la de aplicación).

La mayoría de las funciones del nivel de enlace se implementan en el hardware de los equipos. Esto hace que los protocolos de nivel de enlace se modifiquen poco con el tiempo.

Los tipos de servicio que la capa de enlace puede suministrar a la capa de red son normalmente los siguientes:

- **Servicio no orientado a conexión y sin acuse de recibo** → Apropiado cuando la tasa de error es muy baja (redes locales, transmitir información en tiempo real y videoconferencia). La comprobación de corrección de los datos se deja a las capas superiores (normalmente al nivel de transporte).
- **Servicio no orientado a conexión con acuse de recibo** → Se produce un acuse de recibo para cada trama enviada a fin de que el emisor se asegure que la trama llega al destino. Suele utilizarse en redes con tasas de error más altas, por ejemplo, en redes inalámbricas.
- **Servicio orientado a conexión con acuse de recibo** → Es el servicio más seguro. El emisor y el receptor establecen una conexión explícita de antemano. Las tramas a enviar se numeran y tanto emisor, como receptor, se aseguran de que todas son recibidas correctamente en el destino y transmitidas a la capa de red una sola vez.

En el servicio orientado a conexión se pueden distinguir tres fases:

- Establecimiento de la conexión.
- Envío y recepción de los datos.
- Cierre o terminación de la conexión.

5. Introducción a Ethernet.

Para realizar las prácticas nos basaremos en Ethernet, la tecnología LAN más utilizada, la cual opera en la capa de enlace y en la capa física.
Ethernet proporciona una forma estándar de conectar equipos a la red a través de una conexión por cable. Se utiliza para redes de área local. Ethernet envía los datos en forma de paquetes o tramas. Estas tramas incluyen direcciones de origen y destino y mecanismos de detección de errores.

Los estándares de Ethernet definen los protocolos de la capa de enlace y las tecnologías de la capa física. Actúan en dos subcapas separadas de la capa de enlace de datos: capa de control de enlace lógico (LLC) y capa MAC (figura 3).

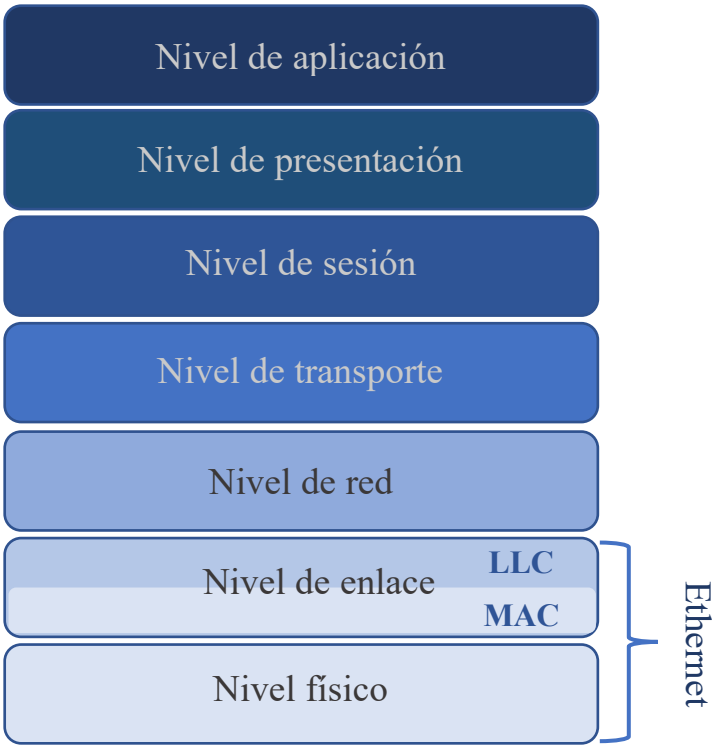

Figura 3: Subcapas de nivel de enlace.

La capa de control de enlace lógico (LLC):

- Maneja la comunicación entre las capas superiores e inferiores.
- Toma los datos del protocolo de red y agrega información de control para ayudar a entregar el paquete al destino.

La capa MAC:

- Constituye la subcapa inferior de la capa de enlace de datos.
- Se implementa mediante hardware, por lo general en la NIC (Network Interface Controller), tarjeta de interfaz de red del equipo.

Funciones de la capa MAC:

- Encapsulación de datos (funciones):
  - Delimitación de tramas: identifica un grupo de bits que componen una trama; sincronización entre los nodos emisor y receptor.
  - Direccionamiento: cada encabezado Ethernet que se agrega a la trama, contiene la dirección física (dirección MAC) que permite que la trama se entregue a un nodo de destino.
  - Detección de errores: cada trama de Ethernet contiene una cola con una comprobación de redundancia cíclica (CRC) del contenido de la trama.

- Control de acceso al medio:
  - Responsable de la ubicación de tramas en los medios.
  - Se comunica directamente con la capa física.
  - Si hay varios dispositivos en un único medio que intentan reenviar datos simultáneamente, los datos colisionan. Ethernet proporciona un método para controlar la forma en que los dispositivos comparten el acceso mediante el uso de una tecnología de acceso múltiple por detección de portadora (CSMA).

6. Formato de tramas Ethernet.

Existen diferentes tipos de tramas Ethernet (Ethernet original y el estándar IEEE 802.3). Las tramas Ethernet tienen una longitud mínima de **64 bytes** y una longitud máxima de **1518 bytes** (sin incluir el preámbulo ni el delimitador de inicio de trama).

El formato de la trama Ethernet más usado actualmente sería el siguiente:

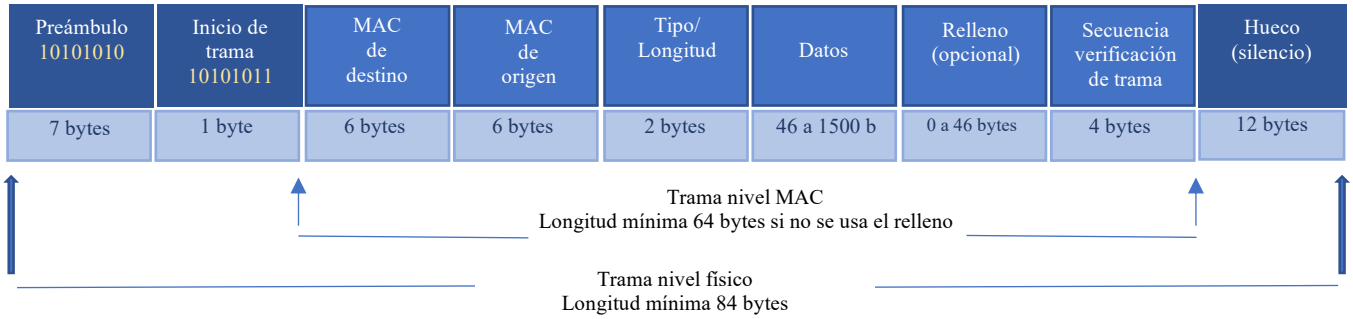

Figura 4: Campos de una trama Ethernet (Ethernet original).

Preámbulo: patrón que alterna unos y ceros utilizado para la sincronización entre emisor y receptor (patrón de bits 10101010 repetido 7 veces). Se usa en implementaciones de Ethernet de 10 Mbps o menores. Las versiones más veloces de Ethernet son síncronas y esta información de temporización es redundante, pero se retiene por cuestiones de compatibilidad. Esta información es procesada por el hardware de NIC.

Inicio de Trama: Se utiliza para delimitar el inicio de la trama Ethernet. Usa el patrón 10101011.

Básicamente, el preámbulo y el delimitador de inicio de trama, indican al receptor que se prepare para recibir una trama nueva.

MAC de destino: contiene la dirección de MAC destino.
La dirección destino puede ser **unicast** (un solo destino), **multicast** (múltiples destinos) o de **broadcast** (todos los destinos) (figura 5).

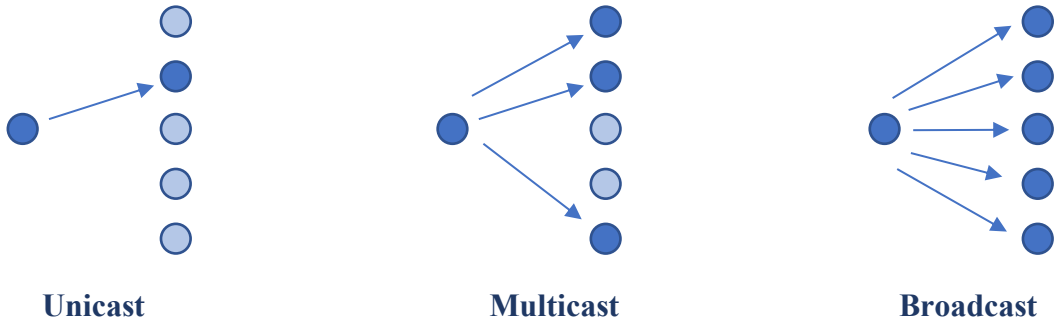

Figura 5: Tipos de direcciones.

MAC de origen: contiene la dirección de MAC origen. La dirección origen generalmente es la dirección unicast. Sin embargo, existe un número creciente de protocolos virtuales en uso que utilizan y a veces comparten una dirección MAC origen específica para identificar la entidad virtual.

Cada dirección MAC origen o destino, tiene una longitud de 48 bits, o seis octetos, expresada como 12 dígitos hexadecimales, 0-9, A-F. Un formato común sería 12:34:56:78:9A:BC.
Los primeros seis números hexadecimales indican el fabricante de la tarjeta de interfaz de red y los seis restantes corresponden al número de serie de la NIC.

Tipo: se define como un campo longitud o tipo. Si el valor es:

- > 0x0600 (> 1536 en decimal) → Está indicando el **tipo** de trama (ej: 0x0800: IPv4; 0x0806: ARP).
- < 0x0600 (< 1536 en decimal) → Está indicando la **longitud** del campo datos.

La Ethernet original usa este campo como el campo tipo, para definir el protocolo de nivel superior encapsulado en la trama Ethernet (ejemplo: 0x800 para IPv4, 0x86DD para IPv6, 0x806 para ARP, …). Esta trama será la que usemos nosotros en las prácticas.

Datos: este campo transporta los datos encapsulados por los protocolos de nivel superior. Tiene un mínimo de 46 y un máximo de 1500 bytes. Cuando no se rellena con el mínimo de bytes, se inserta un relleno no especificado inmediatamente después de los datos para que la trama cumpla con la longitud mínima especificada.

Secuencia de verificación de trama (FCS): es utilizada por la NIC para identificar errores durante la transmisión. Contiene un valor de verificación CRC creado por el dispositivo emisor (abarca las direcciones de trama, tipo y campo de datos) El receptor lo recalcula para verificar la existencia de tramas dañadas.

Hueco: Es el silencio que marca el final de la trama. Su duración es de 9,6 μs, equivalente a 12 bytes. Cuando una estación desea transmitir y detecta el medio libre, debe esperar ese tiempo antes de comenzar para garantizar el hueco mínimo entre tramas. Si esto no fuera así, la trama podría comenzar justo después de que la otra terminara, por lo que se podría interpretar erróneamente como continuación de la trama en curso.

7. Software de trabajo.

7.1 Ubuntu 20.0.4. para trabajar en modo gráfico.

Estas prácticas se van a realizar sobre una distribución GNU/Linux para poder acceder a las capas internas del núcleo del sistema operativo y hacer uso de las interfaces de red disponibles. Para la realización de estas prácticas se necesitarán los siguientes paquetes:

- Paquete net-tools
- Paquetes de desarrollo, build-essential (Ubuntu/Debian) o @development-tools @development-libraries (Fedora)
- Librería libpcap-dev
- Librería libpthreads
- Visual Studio Code

La distribución sugerida para la realización de las prácticas será Ubuntu 20.04 y será para la que se realice la instalación del entorno de trabajo. Además, se dispondrá de una imagen de una máquina virtual con todas las herramientas correctamente instaladas y configuradas.

7.2. Instalación del paquete net-tools.

Este paquete instalará un conjunto de herramientas básicas para utilizar las interfaces de red. Para ello habrá que abrir un terminal y ejecutar los siguientes comandos:

```
sudo apt-get update

sudo apt-get install net-tools
```

7.3. Instalación del paquete build-essential

Este paquete instalará los compiladores y debuggers necesarios para compilar la práctica:

```
sudo apt-get install build-essential
```

7.4. Instalación del paquete libpcap-dev

PCAP es una interfaz de una aplicación dedicada a la captura de paquetes. Su librería es open-source y está escrita en C. En los sistemas basados en Unix recibe el nombre de libpcap, mientras que su versión para Windows se denomina WinPcap. Algunas aplicaciones famosas que usan pcap son: Whireshark, Tcpdump, nmap, Cain & Abel y

Snort. En esta práctica vamos a necesitar la librería de desarrollo de libpcap que se conoce como libpcap-dev:

```
sudo apt-get install libpcap-dev
```

7.5. Instalación del paquete Visual Studio Code

Visual Studio Code es un IDE de programación desarrollado por Microsoft y que permite una configuración muy exhaustiva a través de sus ficheros de configuración. Para su instalación haremos uso del gestor de paquetes *snap*. En un terminal deberemos escribir lo siguiente:

```
sudo snap install code --classic
```

Adecuación de Visual Studio Code

Una vez que todas las librerías están instaladas, así como el IDE de trabajo, vamos a adecuar el entorno para la realización de las prácticas. Para poder realizar las prácticas, es necesario tener permisos de administrador, es por esto que vamos a crear un script que se llame **gdb** que permita ejecutar la aplicación desde el IDE con permisos de superusuario. En primer lugar, abrimos un terminal y creamos el script:

```
sudo nano /usr/local/bin/gdb
```

Una vez se ha abierto el editor, se ha de escribir lo siguiente:

```
sudo /usr/bin/gdb "$@"
```

Para finalizar la edición del documento, una vez escrito el comando se pulsará control + x y se responderá sí. Por último, se cambiarán los permisos del pequeño script que acabamos de crear:

```
sudo chmod +x /usr/local/bin/gdb
```

Para que no pida la contraseña cada vez que se ejecuta la aplicación se debe introducir el siguiente comando en el fichero sudoers.

Usando el siguiente comando se puede editar el fichero:

```
sudo visudo
```

Y se debe insertar la siguiente línea

```
NOMBRE_USUARIO_LINUX ALL = NOPASSWD: /usr/bin/gdb
```

El **NOMBRE_USUARIO_LINUX** será el usuario de Linux que uséis.

Una vez realizado ese paso hay que reiniciar la máquina para que funcione correctamente. Por ultimo hay que modificar el fichero *launch.json*, modificando la entrada *"miDebuggerPath":*

```
"miDebuggerPath": "/usr/local/bin/gdb"
```

Instalar los fuentes de la librería glibc:

```
sudo apt install glibc-source
```

Comprobamos la versión de glibc que vamos a instalar:

```
cd /usr/src/glibc
```

```
ls
```

```
@ubuntu:/usr/src/glibc$ ls
ian   glibc-2.31   glibc-2.31.tar.xz
```

Descomprimimos el archivo glibc de la versión correspondiente:

```
sudo tar xvf glibc-2.31.tar.xz
```

Posteriormente, en Visual Studio editamos el fichero *launch.json*, añadiendo la siguiente información en cualquier parte del apartado de *"configurations": {*

"sourceFileMap": {
"/build/glibc-ZN95T4": "/usr/src/glibc"
},

Debemos tener cuidado con el código que introducimos (ZN95T4), puede cambiar con cada máquina. Si hay algún problema, debemos comprobar el código que se muestra en el error de glibc y éste será el que debemos insertar en el fichero launch.json.

Para, finalmente, adecuar el IDE se deben instalar los conectores entre el IDE y las herramientas de compilación de C/C++ (figura 6).

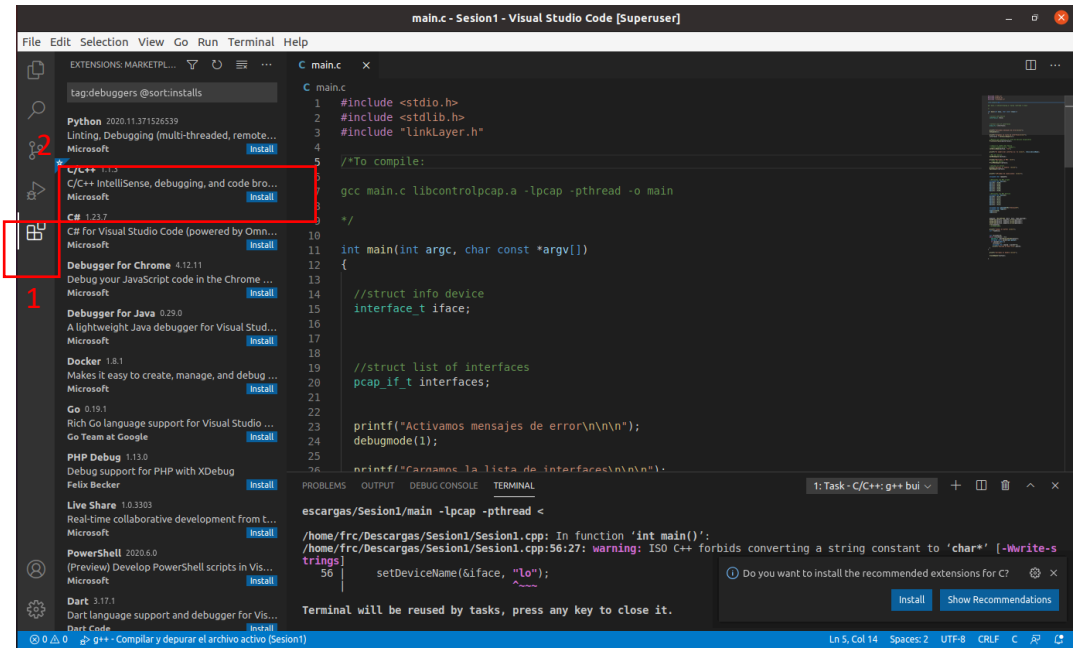

Figura 6: Configuración de Visual Studio Code

Para ello pulsaremos sobre el botón de extensiones (marcado en la figura como 1) e instalaremos el paquete de C/C++ (marcado como 2).

7.6. Instalación del wireshark

Una de las herramientas que usaremos a la hora de realizar la práctica será el analizador de protocolos Wireshark. Esta herramienta permite capturar paquetes de la tarjeta de red y analizarlos en su interfaz. Para su instalación se deberá ejecutar el siguiente comando en el terminal:

```
sudo apt-get install wireshark
```

En el medio de la instalación pedirá configurar los permisos para que usuarios que no sean administradores puedan capturar paquetes. En este caso responderemos que *sí*.

8. Explicación de la librería LinkLayer.

La librería está formada por tres ficheros ".h" y un fichero ".a". Los ficheros .h son las cabeceras de las funciones contenidas en la librería.

Kbhit.h

Es una librería para la gestión del teclado, incluye las funciones:

int getch(void): Esta función captura caracteres que provienen del teclado. Una vez invocada la función no devolverá el control a la aplicación hasta que se haya pulsado cualquier tecla.

int kbhit(void): La función kbhit comprueba si se ha pulsado una tecla. En caso afirmativo, el valor de la tecla pulsada puede ser recogida con la función **getch.**

Lista.h

La librería permite utilizar una lista enlazada para almacenar paquetes del tipo ***assembled_Packet*** (cuyo alias es *apacket_t*) (figura 7)**.** Siendo la estructura de los paquetes:

```
typedef struct assembled_Packet {
    struct pkthdr header;
    const unsigned char *packet;
}apacket_t;
```

Estos paquetes almacenan la información capturadas por la librería PCAP, en forma de registros con dos campos. El primer campo, de tipo ***pcap_pkthdr***, es la cabecera que pone PCAP a cada uno de los paquetes que captura de la tarjeta de red. La estructura que sigue este campo es:

```
struct pkthdr{
    int caplen; /* length of portion present */
    int len; ;      /* length this packet (off wire) */
    struct timeval ts; /* time stamp */
};
```

Siendo el campo **ts** la marca de tiempo cuando se ha capturado, **caplen** la cantidad de información capturada por PCAP y **len** el tamaño del paquete en cuestión. Los principales campos que utilizaremos en las prácticas serán **len** y **ts**.

El segundo campo del tipo *const unsigned char* * es la información del paquete capturado.

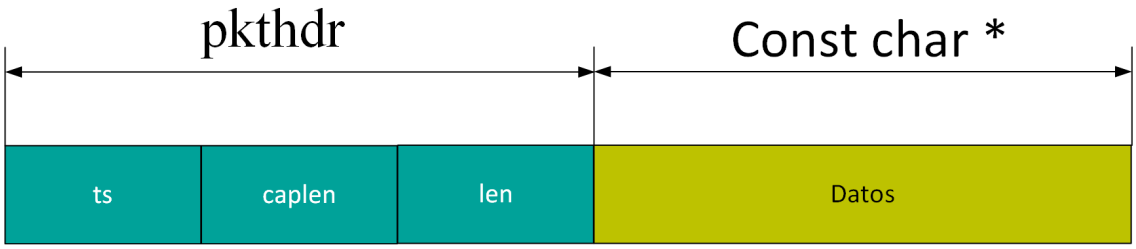

Figura 7: Estructura del assembled_Packet

Las funciones para la gestión de la lista serán las siguientes:

void insertar(unsigned char* x): Esta función inserta la información del paquete en la lista.
int vacia(): Devuelve si la lista está vacía.
apacket_t extraer(): Extrae el primer paquete de la lista
void imprimir(): Imprime todos los paquetes
void liberar(): Elimina todos los paquetes que hay en la lista.

Por último, la librería LinkLayer tiene todas las herramientas necesarias para realizar la práctica. En primer lugar, la librería define la estructura interfaz que será la estructura que gestione la tarjeta de red.

```
typedef struct interface {
    char deviceName[10];
    unsigned char MACaddr[6];
    pcap_t *handle;

    //Handle's stadistics:
    int typeValue;
    unsigned int packetsPassed;
    unsigned int packetsNotPassed;

    //size buffer
    int buffsize;
}interface_t;
```

Con este tipo definido *interface_t* del tipo interface se definirán todas las variables que hagan uso de las interfaces de red. La lista de métodos que contiene la liberaría son los siguientes:

***pcap_if_t GetAvailAdapters()**:* Obtiene la lista de dispositivos y la retorna en una variable del tipo pcap_if_t. Este tipo definido tiene la siguiente estructura:

```
struct pcap_if {
    struct pcap_if *next; /*Puntero al siguiente nodo de la lista*/
    char *name;     /* Nombre de la interfaz */
    char *description;  /* Descripción de la nterfaz, o NULL */
    struct pcap_addr *addresses; /*Direcciones IP de la interfaz*/
    bpf_u_int32 flags;  /* Flags definidas por PCAP_IF_ interface*/
};
typedef struct pcap_if pcap_if_t; /*Tipos definidos del struct*/
```

int setDeviceName(interface_t *iface, char *name): Seleccionar el dispositivo por el nombre de la interfaz.
int GetMACAdapter(interface_t *iface): Obtener la dirección MAC de la interfaz que se pasa por parámetros. La información se almacenará en *interface_t.MACaddr*
int PrintMACAdapter(interface_t *iface): Imprime por pantalla la dirección MAC de la interfaz pasada por parámetros.
int PrintInterfaces(pcap_if_t *interfaces): Muestra todas las interfaces disponibles.
int OpenAdapter(interface_t *iface): Pone el adaptador en modo promiscuo y comienza a capturar paquetes.
int CloseAdapter(interface_t *iface): El adaptador deja de estar en modo promiscuo y no capturará más paquetes llegados de la red.
apacket_t ReceiveFrame(interface_t *iface): Recibir una trama del buffer de almacenamiento.
int SendFrame(interface_t *iface, unsigned char *p, int payloadSize): Enviar una trama a la red.
unsigned char * BuildFrame(unsigned char *srcMAC, unsigned char *dstMAC, unsigned char* protocol, unsigned char *payload): Construye una trama, para ello es necesario disponer de la dirección origen, el destino, el protocolo/tamaño de la trama y la información que debe contener la trama Ethernet (también conocido como payload).
unsigned char * BuildHeader(unsigned char *srcMAC, unsigned char *dstMAC, unsigned char* protocol): Construye una trama, al igual que la función anterior, pero sólo crea la cabecera de la trama, es decir, la trama resultante sólo tendrá: MAC Destino | MAC Origen | Tipo
void FreeBuffer(): Libera el buffer de cualquier información que haya podido recibir con anterioridad.

Configuración del fichero task.json

Para que funcione correctamente la aplicación hay que configurar el proyecto de forma que compile la librería LinkLayer y el resto de requisitos que se han instalado anteriormente. Para ello se debe modificar el fichero **tasks.json** que se encuentra en el menú de nuestro proyecto dentro de la carpeta **.vscode** (está marcada en la figura 8 como número 1).

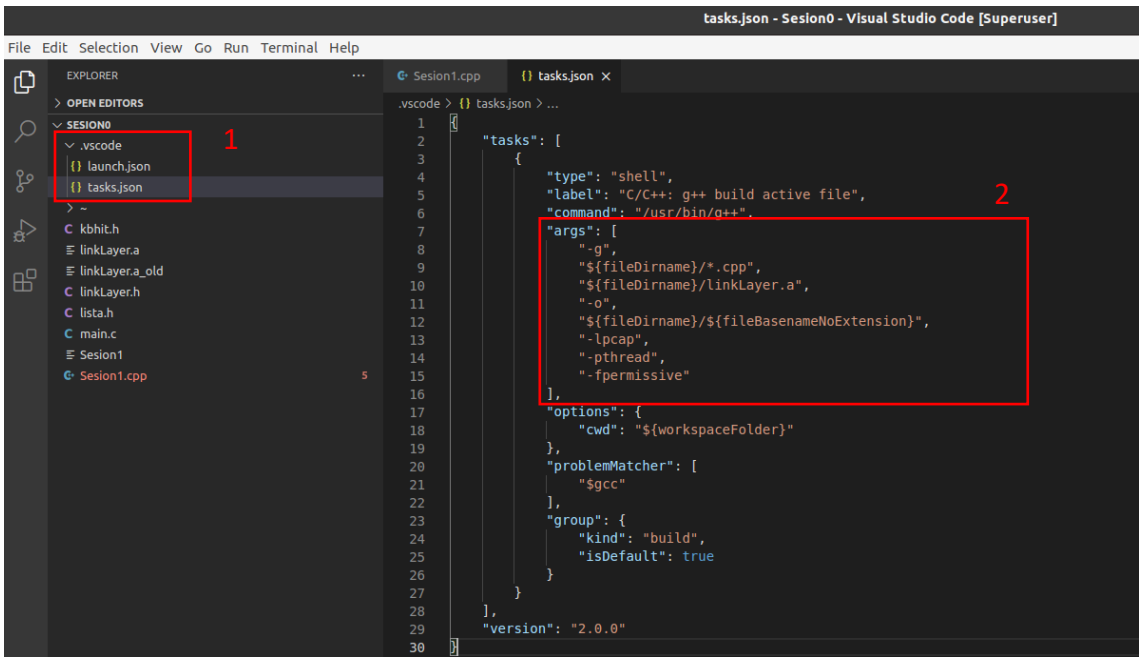

Figura 8: Configuración del proyecto con las librerías

Es obligatorio que en el campo **args** aparezcan tanto las librerías **lpcap**, **pthread** como **fpermissive**, así como la libería **LinkLayer.a**.

9. Tarea a realizar.

Partiendo del ejemplo que se encuentra en el campus virtual (***sesion0***), desarrollar una aplicación que permita al usuario mostrar las interfaces disponibles, elegir qué interfaz se va a usar, extraer la MAC de esa interfaz, almacenarla en una variable del programa principal y mostrarla por pantalla.

10. Salida en pantalla.

La salida en pantalla (figura 9) de la tarea a realizar sería similar a la mostrada a continuación; dependerá de las interfaces disponibles de cada equipo.

```
Interfaces disponibles:
[0] enp0s5
[1] lo
[2] any
[3] bluetooth-monitor
[4] nflog
[5] nfqueue
Seleccione interfaz: 0
Interfaz Elegida: enp0s5
La MAC es: 0:1C:42:DA:D1:45
```

Figura 9: Resultado de la ejecución

11. Cómo ejecutar la práctica.

La práctica se puede ejecutar desde el Visual Studio o directamente desde el Terminal, tal y como se muestra a continuación:

- Visual Studio Code: Opción ***Run/Run Without Debuggin***
- Terminal: ***sudo ./Sesion1*** (Previamente debemos habernos introducido en la carpeta donde se encuentra el fichero ejecutable).

12. Entrega de la sesión práctica.

A través de la herramienta AVUEx, según las instrucciones dadas en clase a este respecto, debe entregarse un archivo comprimido en formato ZIP o RAR que contenga lo siguiente:

- Un archivo AUTORES.TXT, que incluya nombre y apellidos, por este orden, de los autores de la práctica, así como el grupo al que pertenecen ambos.
- Los archivos fuente de la práctica. **Cada fichero fuente** debe incluir **obligatoriamente** el nombre, apellidos y grupo de los autores de la práctica.
- El fichero ejecutable (compilado a partir de los ficheros fuentes entregados) de la práctica.
- Debe incorporarse documentación interna adecuada y suficiente como para seguir adecuadamente el código.
- La fecha tope para subir esta ***entrega 1*** se comentará en clase. **Cualquier práctica entregada con posterioridad a la fecha y hora indicada se considerará no válida a todos los efectos**.