

PRÁCTICA 2

DETECCIÓN Y ANÁLISIS DE TRAMAS MEDIANTE WIRESHARK.



Prácticas de Fundamentos de Redes y Comunicaciones – Curso 2023/2024

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

19 de febrero de 2024

Índice

1. Objetivos.
2. Uso de Wireshark.
3. Ejemplo de envío y captura de paquetes.
 - 3.1. Captura de un paquete o trama mediante Wireshark.
 - 3.2. Envío de una trama mediante Visual Studio Code.
4. Información de tarjeta de red (ifconfig).
5. Explicación de las funciones de la librería para gestión de interfaces, envío y recepción.
 - 5.1. Envío de caracteres.
 - 5.2. Recepción de caracteres.
6. Funciones kbhit y getch.
7. **Tarea a realizar.**
8. **Salida en pantalla.**
9. **Cómo ejecutar la práctica.**
10. Entrega de la sesión práctica.

1. Objetivos.

El principal objetivo de esta entrega es implementar y probar el envío y recepción de caracteres organizados en tramas ethernet a través de las interfaces del equipo. Adicionalmente, se usará el Wireshark para capturar paquetes y analizar las tramas ethernet recibidas, comprobando la información que posee cada uno de los campos que la forman.

2. Uso de Wireshark.

Wireshark es un analizador de protocolos *open-source* que permite inspeccionar protocolos de capa física, de enlace, protocolos de red, capa de transporte y también de

capa de aplicación. Mediante Wireshark podemos realizar capturas de paquetes (entrantes y salientes de la tarjeta de red cableada o inalámbrica) en tiempo real y analizarlos posteriormente.

Wireshark implementa una amplia gama de filtros que facilitan diferentes criterios de búsqueda para todos los protocolos soportados, a través de una interfaz sencilla que desglosa por capas cada uno de los paquetes capturados. Igualmente, proporciona visualización de los campos de cada una de las cabeceras y capas que componen los paquetes.

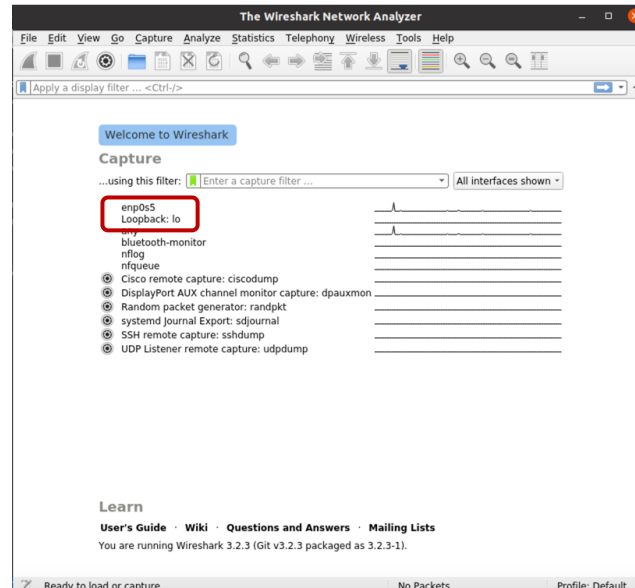


Figura 1: Entorno de Wireshark. Interfaces que se usarán.

Para comenzar a utilizar la aplicación, seleccionaremos la tarjeta de red sobre la que se van a capturar los paquetes entrantes o salientes (figura 1). Para las prácticas usaremos la interfaz correspondiente de la propia tarjeta de red (en la figura 1 viene especificada como **enp0s5**) y la interfaz **loopback**. Ésta última, es una interfaz virtual y no existe físicamente en el equipo, pero realiza todas las funciones de una interfaz normal: siempre está activa, los paquetes destinados a esta interfaz se procesan localmente, los paquetes que se envíen desde esta interfaz a otro destino que no sea la propia interfaz se descartarán sin causar ningún error y no se permite recibir tráfico a través de esta interfaz.

Si no usamos ningún filtro, se muestran todos los paquetes capturados (figura 2). En caso de usar alguno, el filtro no bloqueará la captura de tráfico, solo bloqueará la visibilidad de los paquetes.

Para las prácticas usaremos las opciones principales de Wireshark: comenzar a capturar, dejar de capturar y resetar la captura.

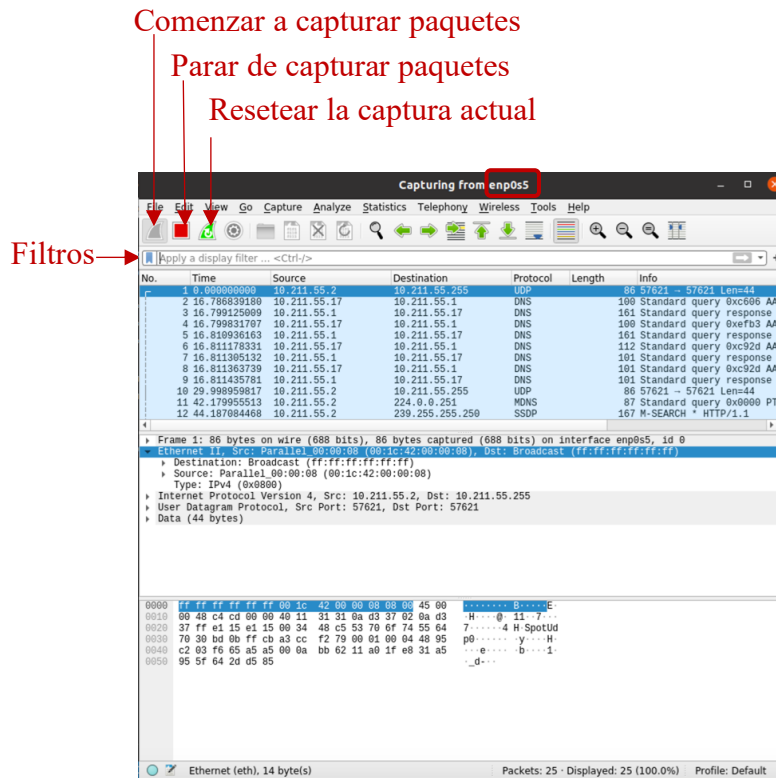


Figura 2: Opciones básicas para capturar paquetes a través de la interfaz enp0s5.

3. Ejemplo de envío y captura de paquetes.

En este apartado vamos a mostrar como enviar una trama que contiene un solo carácter, a través de un programa implementado en C/C++ y lo capturaremos mediante Wireshark a través de la interfaz **loopback**.

3.1. Captura de un paquete o trama mediante Wireshark.

Para capturar paquetes, usaremos Wireshark. Ejecutaremos la aplicación en modo superusuario escribiendo en el terminal **sudo wireshark**. Elegiremos como interfaz **loopback**. Mantendremos el programa abierto para poder capturar paquetes (figura 3). Una vez hayamos enviado el que nosotros deseamos con Visual Studio, analizaremos el paquete capturado.

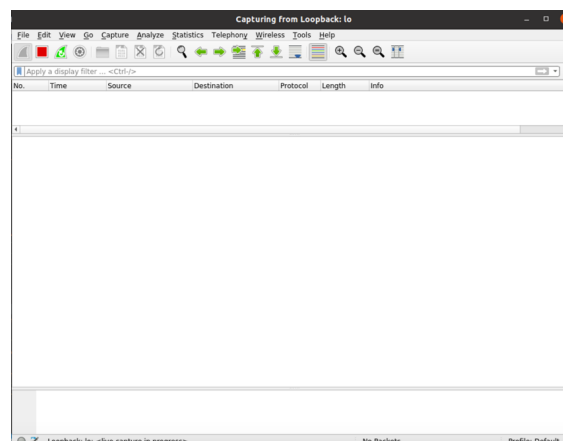
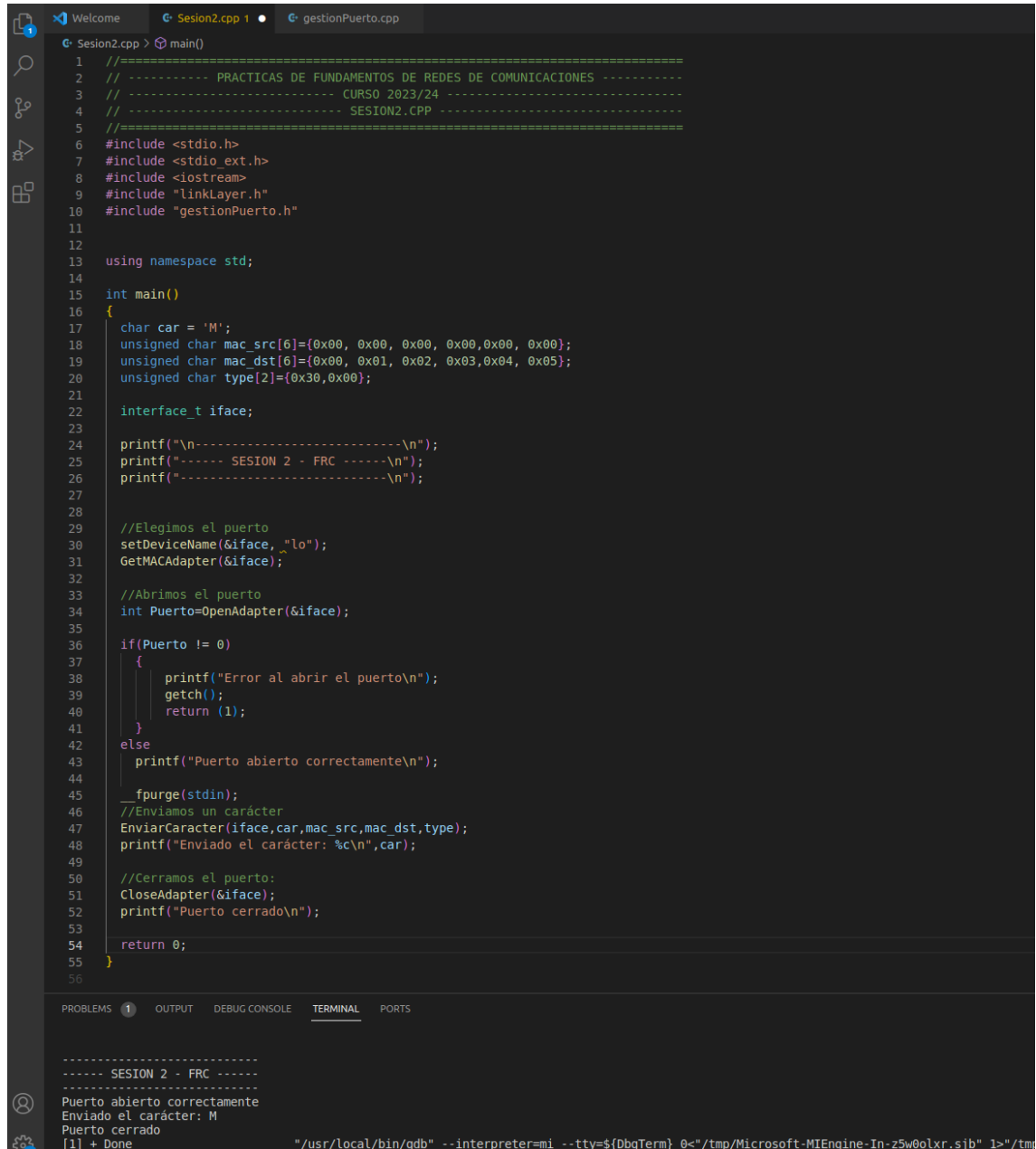


Figura 3: Wireshark listo para capturar paquetes.

3.2. Envío de una trama mediante Visual Studio Code.

Mediante el siguiente programa (Figura 4) enviaremos una trama ethernet que contiene como datos solo el carácter 'M', usando como MAC origen 00:00:00:00:00:00 (mac de **loopback**) y MAC destino 00:01:02:03:04:05. Como tipo de protocolo utilizaremos uno inventado por nosotros, 0x3000.



```

1 //=====
2 // ----- PRACTICAS DE FUNDAMENTOS DE REDES DE COMUNICACIONES -----
3 // ----- CURSO 2023/24 -----
4 // ----- SESION2.CPP -----
5 //=====
6 #include <stdio.h>
7 #include <stdio_ext.h>
8 #include <iostream>
9 #include "linkLayer.h"
10 #include "gestionPuerto.h"
11
12
13 using namespace std;
14
15 int main()
16 {
17     char car = 'M';
18     unsigned char mac_src[6]={0x00, 0x00, 0x00, 0x00,0x00, 0x00};
19     unsigned char mac_dst[6]={0x00, 0x01, 0x02, 0x03,0x04, 0x05};
20     unsigned char type[2]={0x30,0x00};
21
22     interface_t iface;
23
24     printf("\n-----\n");
25     printf("----- SESION 2 - FRC -----\n");
26     printf("-----\n");
27
28     //Elegimos el puerto
29     setDeviceName(&iface, "lo");
30     GetMACAdapter(&iface);
31
32     //Abrimos el puerto
33     int Puerto=OpenAdapter(&iface);
34
35     if(Puerto != 0)
36     {
37         printf("Error al abrir el puerto\n");
38         getch();
39         return (1);
40     }
41     else
42     printf("Puerto abierto correctamente\n");
43
44     __fpurge(stdin);
45     //Enviamos un carácter
46     EnviarCaracter(iface,car,mac_src,mac_dst,type);
47     printf("Enviado el carácter: %c\n",car);
48
49     //Cerramos el puerto:
50     CloseAdapter(&iface);
51     printf("Puerto cerrado\n");
52
53     return 0;
54 }
55
56

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

-----
----- SESION 2 - FRC -----
-----
Puerto abierto correctamente
Enviado el carácter: M
Puerto cerrado
[1] + Done
/usr/local/bin/gdb --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-z5w0lXr.sjb" l>"/tmp/

```

Figura 4: Programa que envía una trama que contiene como datos el carácter 'M'.

Cuando ejecutemos el programa anterior, en wireshark comprobaremos que habremos capturado el paquete deseado, por lo que detendremos la captura de más paquetes para visualizar solo el paquete enviado por nosotros. La información que se reflejará en dicha aplicación se muestra en la figura 5. En dicha figura también se describen los paneles en los que se divide wireshark, así como dónde se muestran los diferentes campos de los que consta un paquete ethernet capturado.

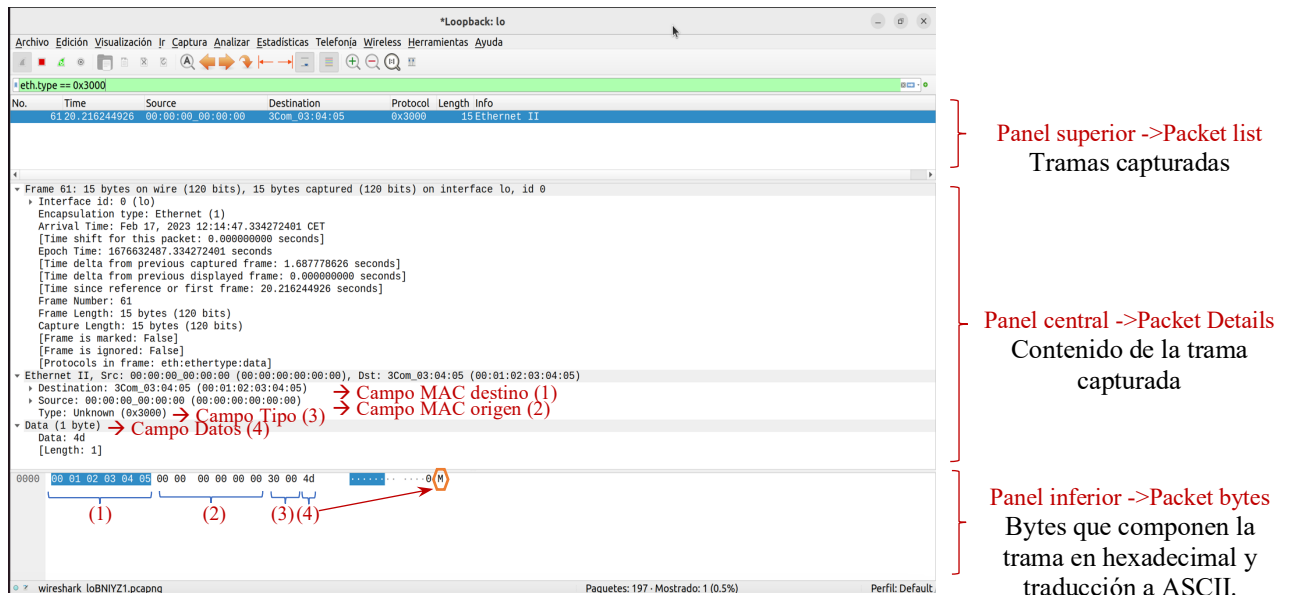


Figura 5: Campos de 1 paquete capturado mediante loopback cuyo dato contiene el carácter 'M'.

4. Información de la tarjeta de red mediante ifconfig.

El comando **ifconfig** permite configurar interfaces de red de un dispositivo, al igual que mostrar información de dichas interfaces usando dicho comando sin parámetros (figura 6). La información que nos interesa a nosotros es la correspondiente al nivel de enlace, es decir, la correspondiente a ethernet. En la figura 6 se muestran bordeadas de un marco rojo, la **mtu** que es la unidad máxima de transferencia, (para ethernet es 1500 bytes) (1), la dirección física (MAC) de una interfaz de red (2), la velocidad de ethernet (en este caso 1 Gb) (3), los paquetes recibidos en bytes (4) y los paquetes enviados en bytes (5).

```

mar@ubuntu: ~
mar@ubuntu:~$ ifconfig
enp0s5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 (1)
    inet 10.211.55.17 netmask 255.255.255.0 broadcast 10.211.55.255
    inet6 fdb2:2c26:f4e4:0:3fdd:4d3a:9c50:5abf prefixlen 64 scopeid 0x0<global>
    inet6 fdb2:2c26:f4e4:0:7cb8:c29c:5440:30be prefixlen 64 scopeid 0x0<global>
    inet6 fe80::ee05:37cd:8d1c:8612 prefixlen 64 scopeid 0x20<link>
    ether 00:1c:42:da:d1:45 txqueuelen 1000 (Ethernet) (2) (3)
    RX packets 9100 bytes 7678002 (7.6 MB) (4)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7718 bytes 1015362 (1.0 MB) (5)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Bucle local)
    RX packets 1474 bytes 182670 (182.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1474 bytes 182670 (182.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

mar@ubuntu:~$

```

Figura 6: Información de tarjetas de red mediante ifconfig.

5. Explicación de las funciones de la librería para gestión de interfaces, envío y recepción.

5.1. Envío de caracteres.

Para el envío de caracteres haremos uso de la librería **linkLayer.h**. A continuación, se muestra el pseudocódigo de cómo podría ser la función **EnviarCaracter**.

```
void EnviarCaracter (interface, datos, mac_origen, mac_destino, tipo)
{
    ReservarMemoriaDatos;
    AlmacenarDatos;
    ConstruirTrama;
    EnviarTrama;
    LiberarMemoriaDatos;
}
```

5.2. Recepción de caracteres.

Igualmente, para la recepción de caracteres haremos uso de la librería **linkLayer.h**. A continuación, se muestra el pseudocódigo de cómo podría ser la función **RecibirCaracter**.

```
char RecibirCaracter (interface)
{
    Trama=RecibirTrama;
    QuedarseCampoCompletoDatosTrama;
    //Acordaos, el campo datos contiene la mac destino, mac origen, tipo y los
    datos propiamente dichos (caracteres) en este orden.
    Si (hay algo en el campo datos)
        QuedarseDatosRecibidos; //Quedarse con los caracteres recibidos.
        DevolverDato;
    Sino
        DevolverValor0;
}
```

6. Funciones kbhit() y getch().

La función **kbhit()** permite detectar la pulsación de una tecla. En caso de pulsar una tecla, dicha función la imprime en pantalla y devuelve un valor distinto de 0. Si no se pulsa ninguna, devolvería 0.

La función **getch()** permite leer una tecla pulsada (por ejemplo, la tecla pulsada y detectada por kbhit()) y la almacenaría en una variable de tipo carácter. La tecla leída no se mostraría en pantalla.

7. Tarea a realizar.

Partiendo de la práctica anterior, desarrollar un programa que envíe cualquier carácter que el usuario escriba por teclado. Tanto en el emisor, como en el receptor se mostrará el carácter enviado y recibido. El programa finalizará solo cuando el usuario pulse la tecla ESC. Igualmente, se deberá poder recibir cualquier carácter que llegue al buffer a través de la interfaz, y éste se mostrará por pantalla. La MAC destino, en esta práctica, será codificada en el programa principal. Para obtener la MAC de origen, se debe hacer uso del código desarrollado en la sesión 1. En pantalla se deberá mostrar tanto la MAC origen, como la MAC destino.

8. Salida en pantalla.

La salida en pantalla (figura 7) de la tarea a realizar sería similar a la mostrada a continuación; dependerá de las interfaces disponibles de cada equipo.

```

-----
----- SESION 2 - FRC -----
-----
Interfaces disponibles:
[0] eth0
[1] eth1
[2] any
[3] lo
[4] bluetooth-monitor
[5] nflog
[6] nfqueue
[7] dbus-system
[8] dbus-session
Seleccione interfaz: 0
Interfaz Elegida: eth0
La MAC origen es: 8A:88:9A:F8:AB:72
La MAC destino es: 2E:8:A5:3F:16:1A
Puerto abierto correctamente
Pulse los caracteres a enviar:
holaRecibido: a
Recibido: d
Recibido: i
Recibido: o
Recibido: s
^[

-----
----- SESION 2 - FRC -----
-----
Interfaces disponibles:
[0] eth0
[1] eth1
[2] any
[3] lo
[4] bluetooth-monitor
[5] nflog
[6] nfqueue
[7] dbus-system
[8] dbus-session
Seleccione interfaz: 1
Interfaz Elegida: eth1
La MAC origen es: 2E:8:A5:3F:16:1A
La MAC destino es: 8A:88:9A:F8:AB:72
Puerto abierto correctamente
Pulse los caracteres a enviar:
Recibido: h
Recibido: o
Recibido: l
Recibido: a
Recibido: a
adios^[
  
```

Figura 7: Resultado de la ejecución mediante dos terminales.

9. 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 ./Sesion2** (Previamente debemos habernos introducido en la carpeta donde se encuentra el fichero ejecutable).

Lo ideal será probar la práctica usando dos equipos conectados a la red local, pero también podremos comprobar su funcionamiento usando un solo equipo. Para ello será preciso abrir dos instancias de la aplicación, bien desde el Visual Studio o bien desde el terminal. La figura 7 muestra la ejecución abriendo dos instancias desde dos terminales utilizando para cada una de ellas una interfaz de red distinta. La figura 8 muestra como

probar la práctica abriendo una instancia desde Visual Studio y otra a través de un terminal abierto en el propio Visual Studio. Esto se haría usando la opción “*Split Terminal*” y ejecutando la aplicación desde dicho terminal mediante **sudo ./Sesion2**. En este caso usaremos la interfaz virtual **loopback**. El ejemplo se muestra a continuación:

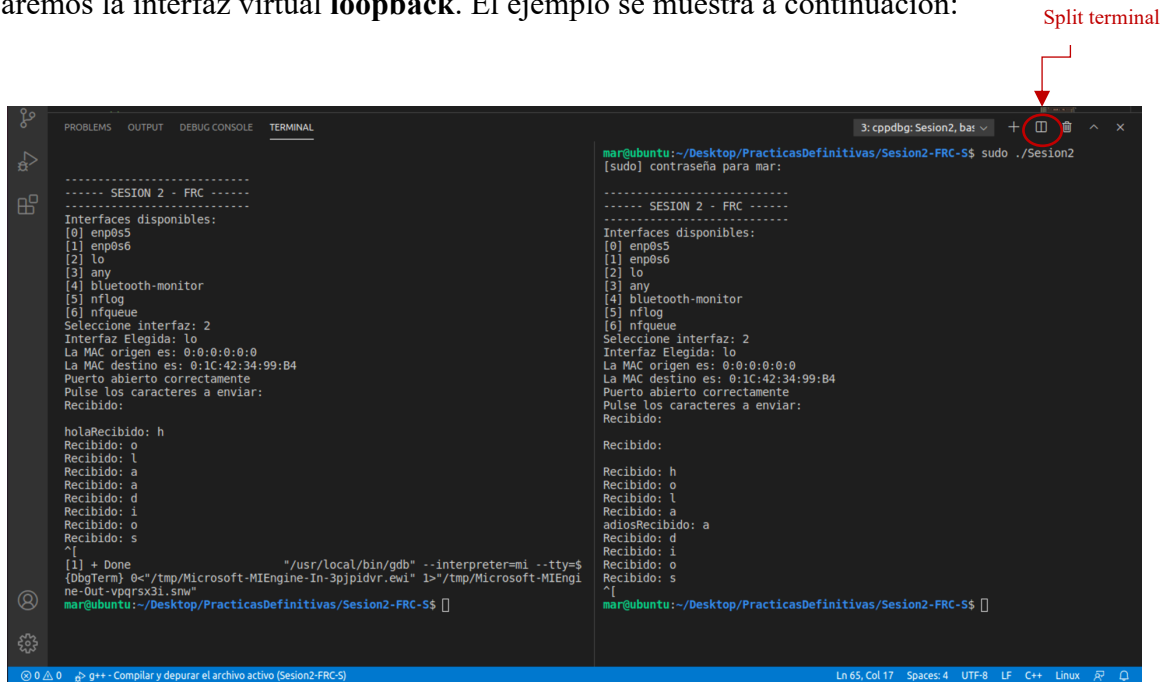


Figura 8: Resultado de la ejecución desde el Visual Studio Code usando opción Split Terminal para la ejecución de dos instancias de la aplicación.

10. 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.
- La carpeta oculta **.vscode**.
- Debe incorporarse documentación interna adecuada y suficiente como para seguir adecuadamente el código.
- La fecha tope para subir esta **entrega 2** se comentará en clase. **Cualquier práctica entregada con posterioridad a la fecha y hora indicada se considerará no válida a todos los efectos.**