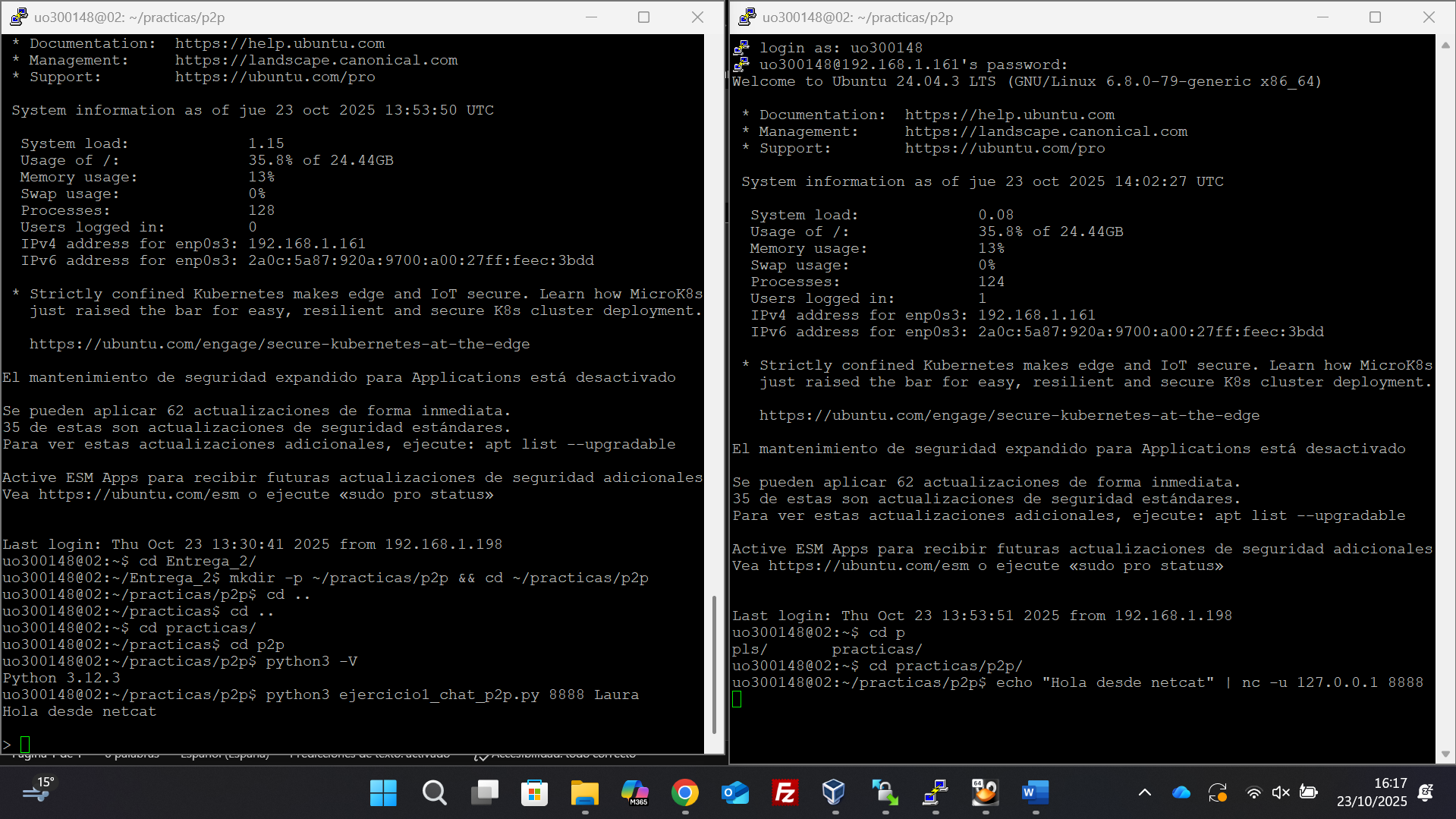
**🧾 Memoria personal – Práctica 1: Implementación de una aplicación de mensajería *peer-to-peer* (UDP)**



**1. Objetivo de la práctica**

El objetivo de esta práctica es diseñar y ejecutar una aplicación de chat simple que permita la comunicación directa entre dos usuarios sin servidor intermedio, utilizando el **protocolo UDP**.  
Con este ejercicio se busca comprender cómo se establece un canal de comunicación entre dos procesos, cómo se envían y reciben datos a través de la red y cómo se gestionan entradas de usuario sin bloquear la ejecución del programa.

**2. Descripción general del programa**

El programa desarrollado se llama **ejercicio1-chat-p2p.py** y permite a dos usuarios chatear entre sí mediante el envío de **datagramas UDP**.  
Cada usuario inicia el programa indicando dos datos:

* El **puerto** que utilizará para escuchar mensajes.
* Su **apodo o nick**, que se antepone automáticamente a todos los mensajes enviados.

Una vez en ejecución, el programa muestra un símbolo de espera (>) y el usuario puede escribir diferentes tipos de comandos:

* El comando **/CHAT <ip> <puerto>** sirve para decirle al programa *a qué dirección y puerto debe enviar los mensajes*.
* El comando **/QUIT** finaliza la comunicación y cierra el programa.
* Cualquier otro texto se interpreta como un **mensaje** que será enviado al destino previamente fijado.

**3. Explicación de funcionamiento (por partes del código)**

**a) Inicio y parámetros**

Cuando se ejecuta el programa, lo primero que hace es **leer los argumentos** introducidos al lanzarlo desde el terminal.  
Estos argumentos indican en qué puerto debe escuchar y qué nombre debe usar el usuario en el chat.  
Si los datos no son válidos, el programa avisa y se cierra.

**b) Creación del socket UDP**

A continuación, el programa crea un **socket UDP**, que es como un “buzón digital” que servirá para enviar y recibir mensajes.  
Después, asocia ese buzón a su número de puerto, de modo que si alguien le envía datos a ese puerto, el programa los recibirá.

UDP no establece conexión como TCP, por lo que los mensajes se envían directamente sin confirmación de entrega.  
Esto significa que el chat funciona más rápido, pero si un mensaje se pierde, no hay aviso.

**c) Bucle principal del programa**

Una vez creado el socket, el programa entra en un **bucle infinito** donde se mantiene escuchando continuamente.  
En cada momento puede ocurrir una de dos cosas:

1. Que el usuario escriba algo en el teclado.
2. Que llegue un mensaje nuevo desde la red.

Para atender a ambas cosas al mismo tiempo, el programa utiliza una función llamada **select()**.  
Esta función vigila simultáneamente el teclado y el socket, y avisa cuando hay algo listo en alguno de los dos, evitando que el programa se bloquee esperando solo una acción.

**d) Recepción de mensajes**

Cuando llega un datagrama UDP desde otro usuario, el programa lo **descodifica** (pasa de bytes a texto) y lo **muestra por pantalla**.  
Después vuelve a mostrar el símbolo > para indicar que puede escribirse un nuevo mensaje.

**e) Envío de mensajes**

Cuando el usuario escribe algo en el teclado, el programa revisa lo que se ha introducido:

* Si empieza por /CHAT, guarda internamente la dirección y el puerto del destinatario.
* Si el texto es normal (por ejemplo, “hola”), lo envía a la dirección guardada, añadiendo delante el nombre del usuario.
* Si lo que se escribe es /QUIT, el programa envía un pequeño aviso de desconexión (si había un destino configurado) y termina.

**f) Cierre del socket**

Al finalizar, el programa **cierra el socket** correctamente para liberar el puerto y evitar errores en futuras ejecuciones.

**4. Pruebas realizadas**

**Prueba individual (dos terminales en la misma máquina)**

* Se abrieron dos terminales, ejecutando el programa con puertos distintos (por ejemplo, 8888 y 8889).
* En cada terminal se estableció el destino del otro mediante el comando /CHAT.
* Se comprobó que los mensajes escritos en una ventana aparecían instantáneamente en la otra.
* Se realizaron capturas de ambas terminales en funcionamiento.

Estas pruebas demostraron que el programa es capaz de enviar y recibir mensajes de forma simultánea en la misma máquina.

**Prueba en red local (con un compañero)**

* Cada estudiante ejecutó el programa en su propio ordenador, usando puertos diferentes.
* Se configuró el destino con la IP del compañero y su puerto correspondiente mediante el comando /CHAT.
* Al enviar mensajes, ambos pudieron verlos correctamente, comprobando la comunicación entre equipos distintos.
* Se tomaron dos capturas adicionales de esta prueba.

**5. Conclusiones**

* Se ha conseguido establecer una comunicación bidireccional simple mediante UDP, sin servidor central.
* El uso de select() permite que el programa lea simultáneamente del teclado y de la red, manteniendo la interacción fluida.
* Se comprobó que UDP no confirma la entrega de mensajes, por lo que en redes inestables podrían perderse datagramas sin aviso.
* La estructura del programa permite entender los conceptos básicos de los **sockets**, **puertos**, y el **modelo cliente-cliente (P2P)**.
* Esta práctica sirve como base para ejercicios más avanzados (como los que usan un *broker* o servidor intermedio).