



**Objetivo**

Mejorar el ejercicio anterior para hacer el intercambio **robusto en TCP**:

* En el **cliente**, garantizar que cada mensaje se envía **completo** usando sendall().
* En el **servidor**, leer **exactamente N bytes (N=5)** con una función recvall(sock, n) que acumula lecturas hasta completar el tamaño esperado.

**Entorno**

* VM Linux de prácticas, mismos directorios del tema.
* Archivos realizados: tcp\_servidor2\_recvall.py (servidor con recvall) y tcp\_cliente2\_sendall.py (cliente con sendall).
* Pruebas locales cliente-servidor en la misma máquina (localhost).

**Procedimiento realizado**

1. **Servidor**: se incorporó la función recvall(sock, 5) para leer el tamaño prefijado. El servidor acepta una conexión, entra en un bucle y, por cada iteración, usa recvall para obtener **justo 5 bytes**; si recibe “FINAL”, cierra la sesión y vuelve a esperar otro cliente.
2. **Cliente**: se sustituyó el envío manual por sendall() para garantizar el envío íntegro de cada bloque. El cliente envía **cinco** mensajes “ABCDE” (5 bytes) y, a continuación, “FINAL”.
3. **Ejecución**: servidor y cliente se lanzan en sesiones separadas; se observa en la consola del servidor la recepción ordenada de los bloques y el cierre tras “FINAL”.

**Resultados observados**

* El servidor muestra “Conectado desde …”.
* Aparecen **cinco** líneas con “Recibido bloque: ABCDE”.
* Después se muestra “Recibido FINAL → cerrar” y el servidor queda de nuevo en “Esperando un cliente…”.
* La comunicación mantiene la **sincronía** incluso si el flujo TCP fracciona envíos o recepciones, porque sendall() asegura el envío completo y recvall() acumula hasta N bytes.

**Evidencias (capturas propuestas)**

* Consola del servidor con: conexión entrante, cinco recepciones de “ABCDE”, recepción de “FINAL” y vuelta al estado de espera.
* Consola del cliente finalizando tras completar los envíos.

**Conclusión**

El uso combinado de **sendall()** (emisor) y **recvall()** (receptor) hace el protocolo de **tamaño prefijado** robusto frente al fraccionamiento natural del stream TCP. Se confirma que el servidor recibe **exactamente 5 bytes por mensaje**, detecta “FINAL” correctamente y queda listo para atender nuevas conexiones.