



FIFO

Sección 1: ¿Qué son los FIFOs?

Explicación teórica

FIFO (First-In, First-Out) es un tipo especial de pipe conocido como *named pipe* (pipe con nombre). A diferencia de los pipes anónimos, los FIFOs existen como archivos especiales en el sistema de archivos y permiten la comunicación entre procesos que **no tienen relación padre-hijo**.

Comparación rápida con pipes anónimos:

Característica	Pipe Anónimo	FIFO (Named Pipe)
Archivo en disco	✗ No	✓ Sí (<code>mkfifo</code>)
Persistencia	✗ Volátil	✓ Hasta que se borra
Comunicación	Solo padre-hijo	Cualquier proceso
Nombre identificable	✗ No	✓ Sí (en el sistema)

Respuesta 2: ¿Por qué un FIFO puede ser usado entre procesos no relacionados?

Porque al estar representado como un **archivo en el sistema de archivos**, **cualquier proceso** que tenga permisos puede abrirlo para leer o escribir, **sin**

necesidad de haber sido creado por el mismo proceso padre.

🔄 Los pipes anónimos, en cambio, existen solo en memoria y **requieren herencia del descriptor de archivo**, por lo que solo sirven entre procesos relacionados (como padre-hijo).

👉 **Respuesta 3: ¿Qué significa el tipo de archivo `p` al listar un FIFO con `ls -l` ?**

La primera letra `p` en los permisos (`prw-r--r--`) significa que es un archivo de tipo **pipe con nombre**, es decir, un **FIFO**.

🧪 Sección 2: Leer y escribir en un FIFO con Python

📖 Teoría rápida

- El FIFO actúa como un **canal unidireccional de comunicación**.
- **Un proceso escribe**, otro **proceso lee**.
- Ambos deben abrir el FIFO, y si no hay un proceso del otro lado, se puede **bloquear** la ejecución.

💻 Ejemplo práctico

Vamos a crear dos scripts: uno que escribe, otro que lee del FIFO.

1. `escritor.py`

```
import os
import time

fifo_path = "canal_chat"

# Asegurarse de que el FIFO existe
if not os.path.exists(fifo_path):
    os.mkfifo(fifo_path)

with open(fifo_path, 'w') as fifo:
```

```
for i in range(5):
    mensaje = f"Mensaje {i}\n"
    print(f"Enviando: {mensaje.strip()}")
    fifo.write(mensaje)
    fifo.flush()
    time.sleep(1)
```

2. lector.py

```
import os

fifo_path = "canal_chat"

# Asegurarse de que el FIFO existe
if not os.path.exists(fifo_path):
    os.mkfifo(fifo_path)

with open(fifo_path, 'r') as fifo:
    print("Esperando mensajes...")
    while True:
        linea = fifo.readline()
        if not linea:
            break
        print(f"Recibido: {linea.strip()}")
```

👉 Respuesta 2: ¿Por qué el `flush()` es importante en el escritor?

En Python, las operaciones de escritura a archivos (incluyendo FIFOs) **usan un búfer**. Eso significa que no se envía el contenido inmediatamente, sino que se acumula hasta que el búfer se llena o se cierra el archivo.

🔄 `flush()` **fuerza a que el contenido se escriba inmediatamente**, permitiendo que el lector lo reciba sin demora.

👉 Respuesta 3: ¿Qué tipo de bloqueo ocurre si un extremo del FIFO no está abierto?

Es un **bloqueo unidireccional**, dependiendo de qué extremo falte:


- Si se abre el FIFO en modo **lectura** y no hay ningún proceso escribiendo, el lector **se bloquea** esperando.
- Si se abre el FIFO en modo **escritura** y no hay ningún lector, el escritor **se bloquea** hasta que alguien lea.

Solo se desbloquean cuando ambos extremos están abiertos. Esto se llama **bloqueo por apertura** del FIFO.

DUDAS Y PREGUNTAS:

¿Qué es un "cursor" en este contexto?


Cuando un proceso abre un archivo o FIFO, se le asigna un **descriptor de archivo**. Ese descriptor **mantiene un "cursor"**, es decir, una **posición actual de lectura o escritura**.

 Imaginá el FIFO como un rollo de papel con mensajes escritos. Cada lector tiene **su propia lupa**, que empieza al principio. Si alguien avanza con su lupa, el otro sigue donde está.

Pregunta 1: ¿Por qué cada lector recibe todos los mensajes?

Porque **cada lector tiene su propio cursor**. No están compartiendo el mismo "punto de lectura".

Cuando el escritor escribe en el FIFO, **el mensaje queda disponible para cualquiera que lea**. Como cada lector empezó desde el inicio, ambos **leen el mismo mensaje**, sin interferirse.

 Si fuera un pipe anónimo **compartido entre procesos**, ahí sí **comparten cursor**. Pero como abrimos el FIFO por separado en cada proceso, cada uno tiene su propio descriptor.

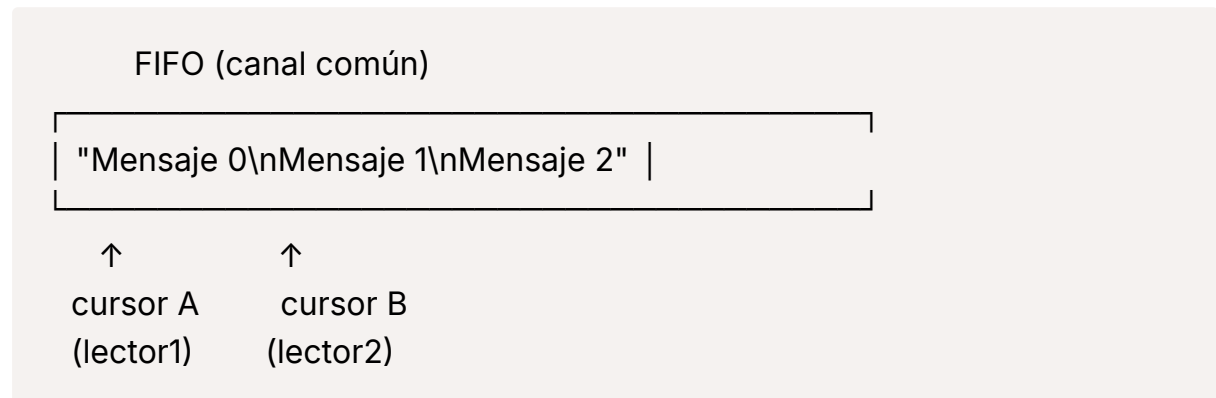
Pregunta 2: ¿Qué significa que el descriptor de archivo no se comparta entre procesos?

Significa que **cada proceso tiene su propia copia del "punto de lectura"**. Si vos y yo abrimos el mismo archivo desde dos programas diferentes, cada uno **mantiene su lugar por separado**. Uno puede estar leyendo la línea 3 y el otro la línea 1.

? Pregunta 3: ¿Qué pasa si un lector se inicia después de que el escritor ya terminó?

Ese lector **no recibe nada** (o sólo los mensajes que queden si el FIFO no fue vaciado), porque el FIFO **no guarda un historial**. Es como llegar tarde a una conversación: **te perdiste lo que ya se dijo**.

🎨 Modelo conceptual: FIFO con dos lectores



✅ Aunque ambos están leyendo **del mismo FIFO**, **cada uno tiene su propia posición de lectura**.

🔄 Proceso

1. **El escritor** mete mensajes al FIFO:

Mensaje 0

Mensaje 1

Mensaje 2

2. **Lector 1** abre el FIFO y empieza a leer desde el principio (cursor A).

3. **Lector 2** también abre el FIFO y empieza desde el principio (cursor B).

📌 **No se pisan** ni se reparten los mensajes: **ambos reciben todos los mensajes**, uno por uno, cada uno por su cuenta.

📌 Comparación rápida

Caso	¿Comparten cursor?	¿Reciben todos los mensajes?
Dos lectores en el mismo FIFO	❌ No	✅ Sí
Dos procesos con un mismo pipe heredado (anónimo)	✅ Sí	❌ No (se reparten)

Conceptos clave

- **Un solo FIFO:** todos los procesos **escriben en él**, y uno solo lo **lee**.
 - Todos los **escritores escriben sin bloquearse**, siempre que haya un lector activo.
 - El **lector lleva el cursor**, así que si no hay lector en el momento justo, **los mensajes pueden perderse**.
-


Resumen final del tema

¿Qué son los FIFOs?

- Son *named pipes*, una forma de comunicación entre procesos basada en archivos especiales.
 - Se crean en disco con `mkfifo`.
 - A diferencia de los pipes anónimos, **persisten** hasta que se eliminan manualmente.
-

¿Cómo se usan?

- Se abren como archivos (`open` , `read` , `write`).
 - Requieren **coordinación**: si no hay lector, el escritor bloquea.
 - Pueden vincular **programas diferentes** fácilmente.
-

 En realidad, los datos **no se "borran" inmediatamente**. Permanecen en el FIFO **hasta que un lector los lea**. Pero si no hay lector activo, el escritor **bloquea** o lanza error, y los datos **no se escriben en absoluto**.