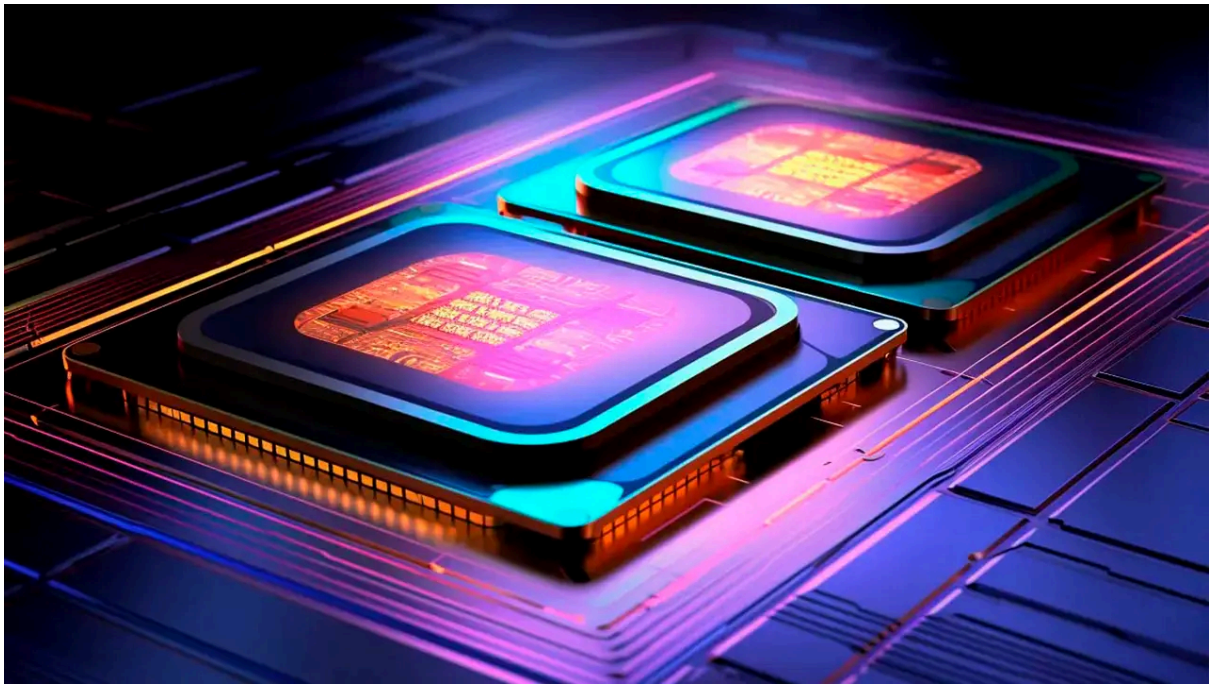


2.4- Comunicación de procesos multi-hilo (linux)

ÍNDICE

1. Script de "Hilos"

Primero, una aclaración importante: Bash en sí mismo no maneja "hilos" (threads) como lo haría un programa en C o Python. Cuando usamos el ampersand (&) en Bash, lo que realmente lanzamos es un sub-proceso (un proceso hijo).

Si ejecutas ps -eLf mientras corre el script, no verás un solo PID con múltiples TIDs (Thread IDs). Lo que verás serán múltiples PIDs, todos "hijos" (con el mismo PPID) del script principal.

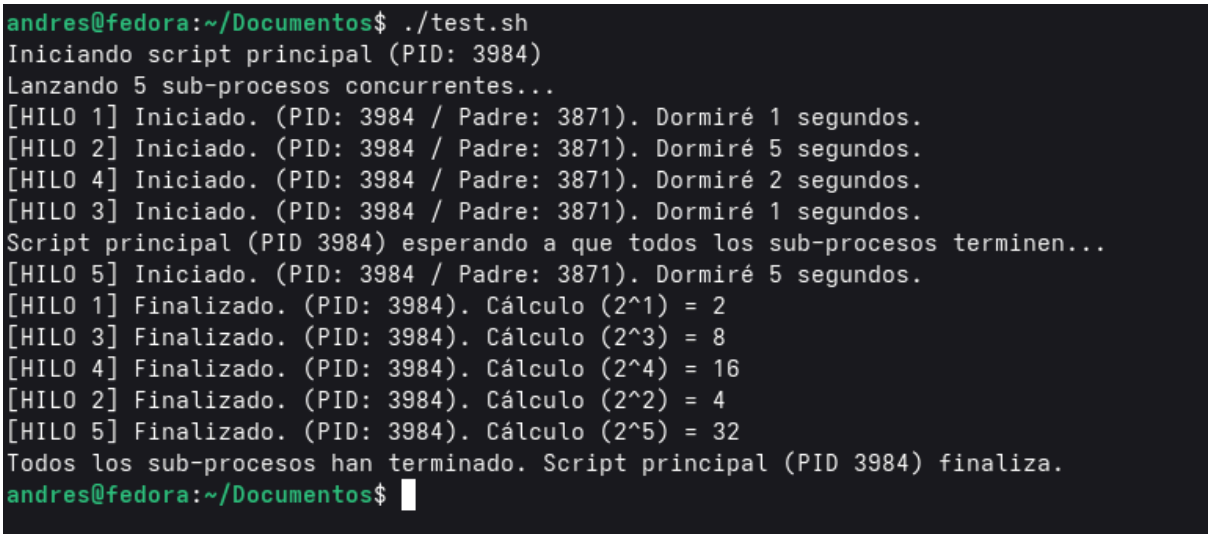
Este script lo demuestra:

```
#!/bin/bash

NUM_PROCESOS=5
SCRIPT_PID=$$
echo "Script principal (PID: $SCRIPT_PID) iniciando."
echo "Lanzando $NUM_PROCESOS sub-procesos..."

# Esta función simula el trabajo
trabajo_hilo() {
    i=$1
    MI_PID=$$
    # Genera un retardo aleatorio entre 1 y 5 segundos
    RETARDO=$((RANDOM % 5 + 1))

    echo "[HILO $i] ¡Iniciado! (Mi PID: $MI_PID / Padre: $PPID).
Voy a dormir $RETARDO seg."
    sleep $RETARDO

    # Un cálculo tonto como ejemplo
    CALCULO=$(echo "2^$i" | bc)
    echo "[HILO $i] ¡Terminé! (PID: $MI_PID). Mi cálculo (2^$i) es
= $CALCULO"
}

# Lanzamos todos los procesos en segundo plano (concurrentes)
for i in $(seq 1 $NUM_PROCESOS); do
    trabajo_hilo $i &
done

echo "Script principal (PID $SCRIPT_PID) esperando a que terminen
los hijos..."

# 'wait' es crucial. El script principal se detiene aquí
# hasta que todos los trabajos en segundo plano (&) hayan
finalizado.
```

```
wait

echo "Todos los sub-procesos terminaron. Script principal (PID
$SCRIPT_PID) finaliza."
```

Cómo probarlo

Guarda el script como hilos.sh y dale permisos: chmod +x hilos.sh
Ejecútalo: ./hilos.sh
Mientras se está ejecutando (tienes unos segundos), abre otra terminal y ejecuta ps -eLf | grep hilos.sh o ps -eLf | grep sleep.
Lo que verás: Verás el proceso ./hilos.sh (ej. PID 5001) y varios procesos sleep cuyo PPID (Parent PID) será 5001. Son procesos distintos, no hilos.



2. Scripts Productor y Consumidor (con mkfifo)

Aquí usaremos un "pipe con nombre" (mkfifo). Es, básicamente, un archivo especial en el sistema de archivos que actúa como un tubo: un script escribe en él y el otro lee. El sistema operativo se encarga de la sincronización.

productor.sh

Este script crea el pipe, registra su PID, envía 5 mensajes y registra su estado al finalizar

```
#!/bin/bash

PIPE_NAME="mi_pipe_ipc"
LOG_FILE="productor.log"
PID=$$

# --- INICIO DE LOG ---
# (Usamos > para sobrescribir el log antiguo)
{
    echo "========================================"
    echo "INICIO SCRIPT PRODUCTOR"
    echo "Fecha/Hora Inicio: $(date)"
    echo "PID: $PID"
    echo "========================================"
} > $LOG_FILE

# Si el pipe no existe (-p), lo creamos
[ -p $PIPE_NAME ] || mkfifo $PIPE_NAME

# 'tee -a' muestra por pantalla Y añade (append) al log
echo "Productor (PID $PID) listo. Enviando a '$PIPE_NAME'..." |
tee -a $LOG_FILE

for i in $(seq 1 5); do
    MENSAJE="Mensaje #$i (desde PID $PID)"
    echo "Enviando: $MENSAJE"

    # Escribimos en el pipe.
    # IMPORTANTE: Esta línea se BLOQUEARÁ si no hay
    # un consumidor leyendo al otro lado.
    echo "$MENSAJE" > $PIPE_NAME

    echo "LOG: Mensaje $i enviado." >> $LOG_FILE
    sleep 1
done

echo "Productor (PID $PID) terminó de enviar." | tee -a $LOG_FILE
```

```
# --- FIN DE LOG (ESTADO Y CONSUMO) ---
# (Usamos >> para añadir al log)
{
    echo "========================================"
    echo "FIN SCRIPT PRODUCTOR"
    echo "Fecha/Hora Fin: $(date)"
    echo "Estado de recursos (PID $PID):"
    # ps nos da una "foto" del consumo del proceso
    ps -p $PID -o pid,ppid,%cpu,%mem,etime,cmd
    echo "========================================"
} >> $LOG_FILE
```

consumidor.sh

Este script registra su PID y se queda escuchando en el pipe. Registra cada mensaje que lee y su estado al finalizar.

```
#!/bin/bash

PIPE_NAME="mi_pipe_ipc"
LOG_FILE="consumidor.log"
PID=$$

# --- INICIO DE LOG ---
{
    echo "========================================"
    echo "INICIO SCRIPT CONSUMIDOR"
    echo "Fecha/Hora Inicio: $(date)"
    echo "PID: $PID"
    echo "========================================"
} > $LOG_FILE

# Nos aseguramos de que el pipe exista
[ -p $PIPE_NAME ] || mkfifo $PIPE_NAME

echo "Consumidor (PID $PID) esperando mensajes en '$PIPE_NAME'..."
| tee -a $LOG_FILE
echo "LOG: Iniciando bucle de lectura." >> $LOG_FILE

# Leemos del pipe línea a línea.
# El bucle 'while read' terminará automáticamente
# cuando el 'productor' cierre el pipe al terminar.
while read -r linea; do
    echo "RECIBIDO: $linea"
    echo "LOG: Recibido '$linea'" >> $LOG_FILE
done < $PIPE_NAME
```

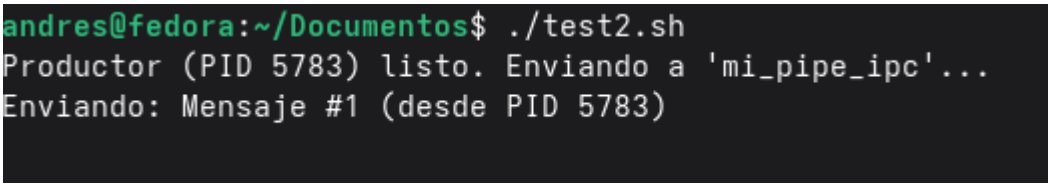
```
echo "Consumidor (PID $PID) finalizado (el pipe se cerró)." | tee
-a $LOG_FILE

# --- FIN DE LOG (ESTADO Y CONSUMO) ---
{
    echo "========================================"
    echo "FIN SCRIPT CONSUMIDOR"
    echo "Fecha/Hora Fin: $(date)"
    echo "Estado de recursos (PID $PID):"
    ps -p $PID -o pid,ppid,%cpu,%mem,etime,cmd
    echo "========================================"
} >> $LOG_FILE
```

3. Cómo probar la comunicación

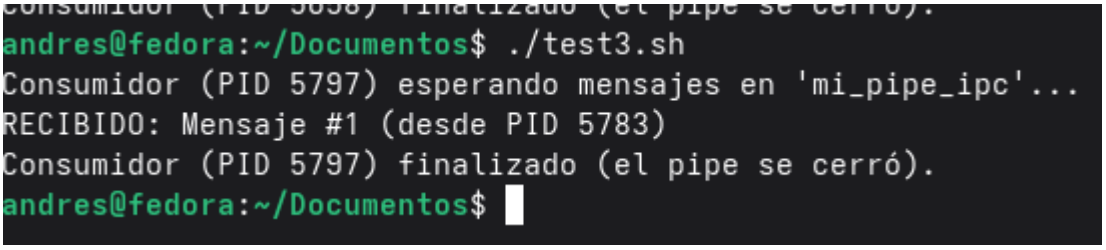
Esto es clave: necesitas dos terminales abiertas al mismo tiempo.

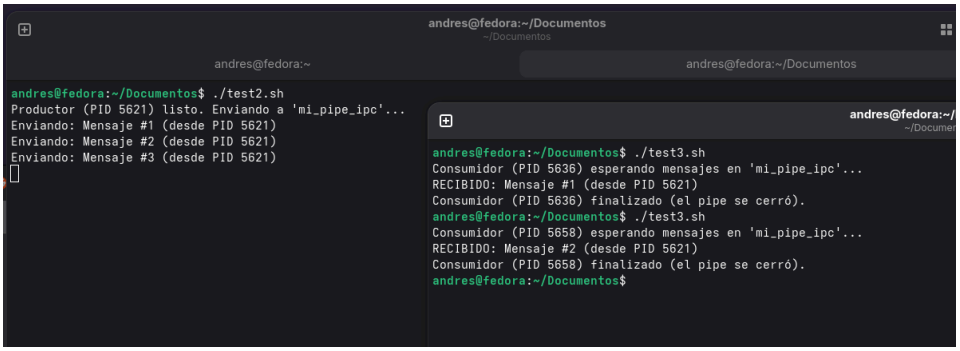
Dale permisos a ambos: `chmod +x productor.sh consumidor.sh`

Terminal 1 (Consumidor): Lanza el consumidor primero. Verás que se queda "pillado" esperando datos. Es normal.



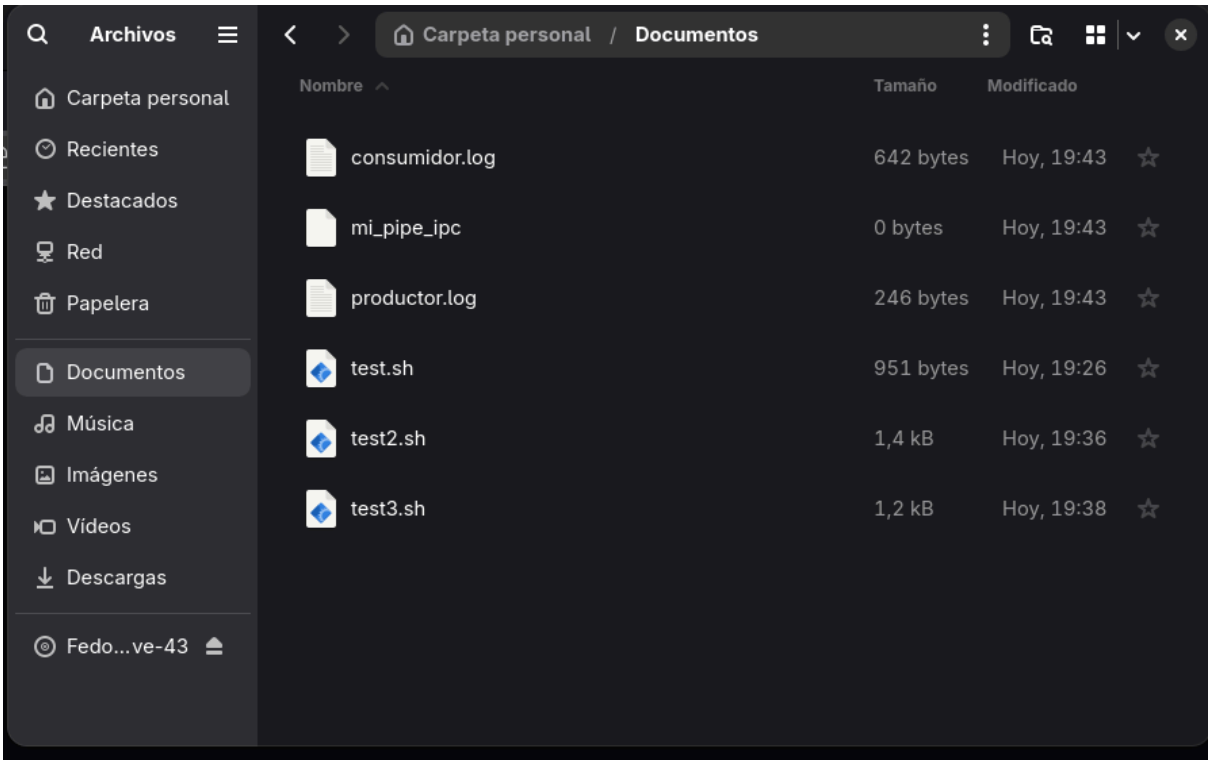
Terminal 2 (Productor): Ahora, en la otra terminal, lanza el productor.

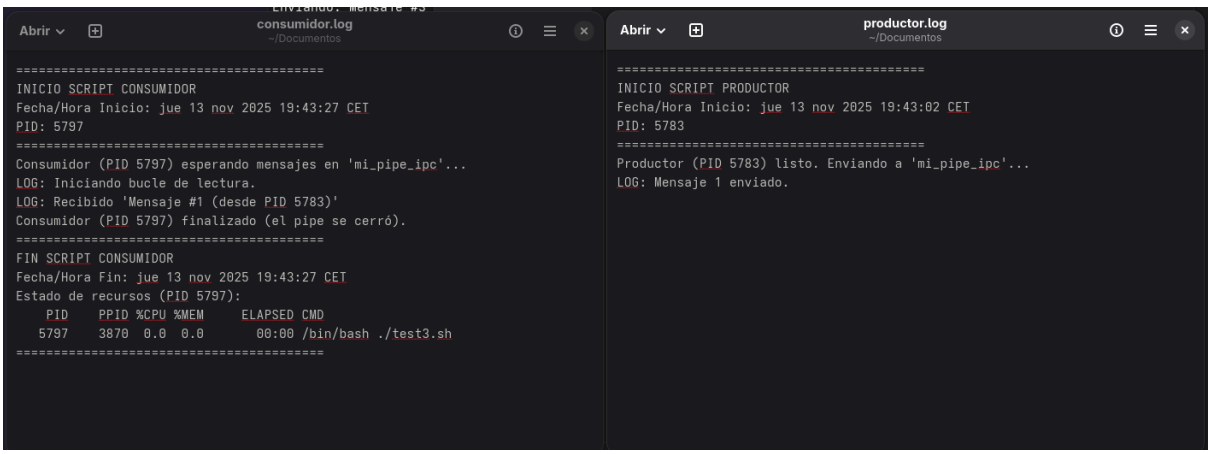




Revisa los Logs

Ahora tendrás dos archivos, productor.log y consumidor.log, que son tu "caja negra" de lo que ha pasado.





4. ¿Para qué sirven estos logs?

Aquí es donde está la utilidad real. Cuando algo falla (y fallará), los logs te dicen qué pasó, cuándo y quién lo hizo.

- Proceso "colgado" (Deadlock):
    - Problema: El consumidor se queda "esperando" y nunca recibe nada.
    - Análisis: Miras productor.log. Si ni siquiera existe, el productor nunca arrancó. Si existe, pero el consumidor.log no dice "RECIBIDO", el pipe está roto o el consumidor falló antes de leer.
    - Problema: El productor se atasca en "Enviando: Mensaje #1...".
    - Análisis: El consumidor.log no existe o murió. El productor está bloqueado (comportamiento normal del pipe) porque nadie está leyendo al otro lado.
- Pérdida de mensajes:
    - Problema: Faltan datos al final del proceso.
    - Análisis: Comparas los logs. Cuentas los "LOG: Mensaje X enviado" en productor.log y los "LOG: Recibido..." en consumidor.log. Si los números no cuadran, sabes exactamente en qué mensaje falló el consumidor.
- Análisis de tiempos (Lentitud):
    - Problema: El sistema va lento.
    - Análisis: Compara las "Fecha/Hora Inicio" y "Fecha/Hora Fin".
        - Si el consumidor empezó a las 19:29:58 y el productor a las 19:30:01, sabes que el consumidor estuvo 3 segundos "idle" (parado), lo cual es normal.
        - Si el productor terminó a las 19:30:06 y el consumidor a las 19:35:00, sabes que el consumidor se quedó 5 minutos haciendo algo después de recibir el último mensaje. El problema está en el consumidor.
- Consumo de recursos:
    - Problema: El servidor se satura cuando se ejecuta el script.
    - Análisis: Miras la sección "Estado de recursos" al final. Si el %CPU o %MEM de uno de los PIDs es altísimo (ej. 95%), has encontrado al culpable.

- Errores de arranque (Orquestación):
    - Problema: El sistema falla aleatoriamente.
    - Análisis: Miras los PIDs y PPIDs (Parent PID). Si ves que el PPID del productor es el PID del consumidor, sabes que el consumidor está lanzando al productor (quizás no querías eso). En nuestro caso, ambos tienen el mismo PPID (ej. 7500, la terminal), así que son "hermanos" independientes.