2.4- COMUNICACIÓN DE PROCESOS MULTI-HILO (LINUX)

Cristóbal Suárez Abad

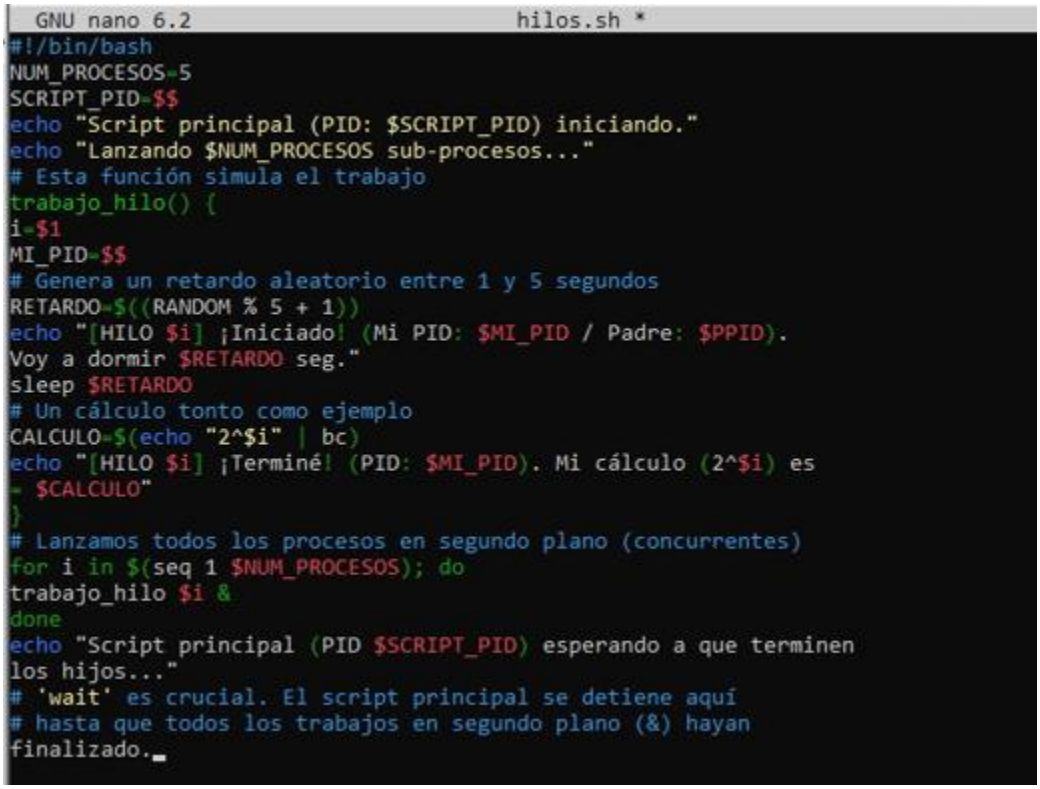
ADMINISTRACIÓN DE SISTEMAS OPERATIVOS - 2º ASIR

Contenido

Crea un script sencillo llamado hilos.sh que lance varios hilos concurrentes.

Cada hilo debe imprimir su número y hacer un pequeño cálculo o retardo (sleep $((RANDOM % 5 + 1))). Observa que todos los hilos pertenecen al mismo proceso (mismo PID) pero tienen TIDs distintos. (ps -eLf o htop)

En Bash no existen hilos reales, pero sí subprocesos concurrentes, cada uno con su propio TID (LWP en Linux). Basta con lanzar funciones en background (&).



```
 GNU nano 6.2                          hilos.sh *
#!/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.
```

```
#!/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."
```

Le damos permisos de ejecución: Abre la terminal y haz que el archivo sea ejecutable:
chmod +x hilos.sh

```
 Tu Nombre  domingo 16 noviembre 2025 09:57
[root@server2asir usuario]$chmod +x hilos.sh
 Tu Nombre  domingo 16 noviembre 2025 09:57
```

Lo ejecutamos:

```
[root@server2asir usuario]$sh hilos.sh
Script principal (PID: 2034) iniciando.
Lanzando 5 sub-procesos...
[HILO 1] ¡Iniciado! (Mi PID: 2034 / Padre: 1818). Voy a dormir 1 seg.
[HILO 2] ¡Iniciado! (Mi PID: 2034 / Padre: 1818). Voy a dormir 1 seg.
Script principal (PID 2034) esperando a que terminen los hijos...
[HILO 4] ¡Iniciado! (Mi PID: 2034 / Padre: 1818). Voy a dormir 1 seg.
[HILO 5] ¡Iniciado! (Mi PID: 2034 / Padre: 1818). Voy a dormir 1 seg.
[HILO 3] ¡Iniciado! (Mi PID: 2034 / Padre: 1818). Voy a dormir 1 seg.
[HILO 2] ¡Terminé! (PID: 2034). Mi cálculo (2^2) es = 4
[HILO 1] ¡Terminé! (PID: 2034). Mi cálculo (2^1) es = 2
[HILO 4] ¡Terminé! (PID: 2034). Mi cálculo (2^4) es = 16
[HILO 3] ¡Terminé! (PID: 2034). Mi cálculo (2^3) es = 8
[HILO 5] ¡Terminé! (PID: 2034). Mi cálculo (2^5) es = 32
Todos los sub-procesos terminaron. Script principal (PID 2034) finaliza.
 Tu Nombre  domingo 16 noviembre 2025 09:59
```

Comprobamos con: **ps -eLf | grep hilos.sh**

```
[usuario@server2asir ~]$ps -eLf | grep hilos.sh
usuario     2062    2021    2062  0    1 09:59 pts/4    00:00:00 grep --color=auto
 hilos.sh
 Tu Nombre  domingo 16 noviembre 2025 09:59
```

Crea dos scripts o programas: uno que envíe mensajes (productor) y otro que los reciba (consumidor).

Configura los scripts para que guarden un log con:

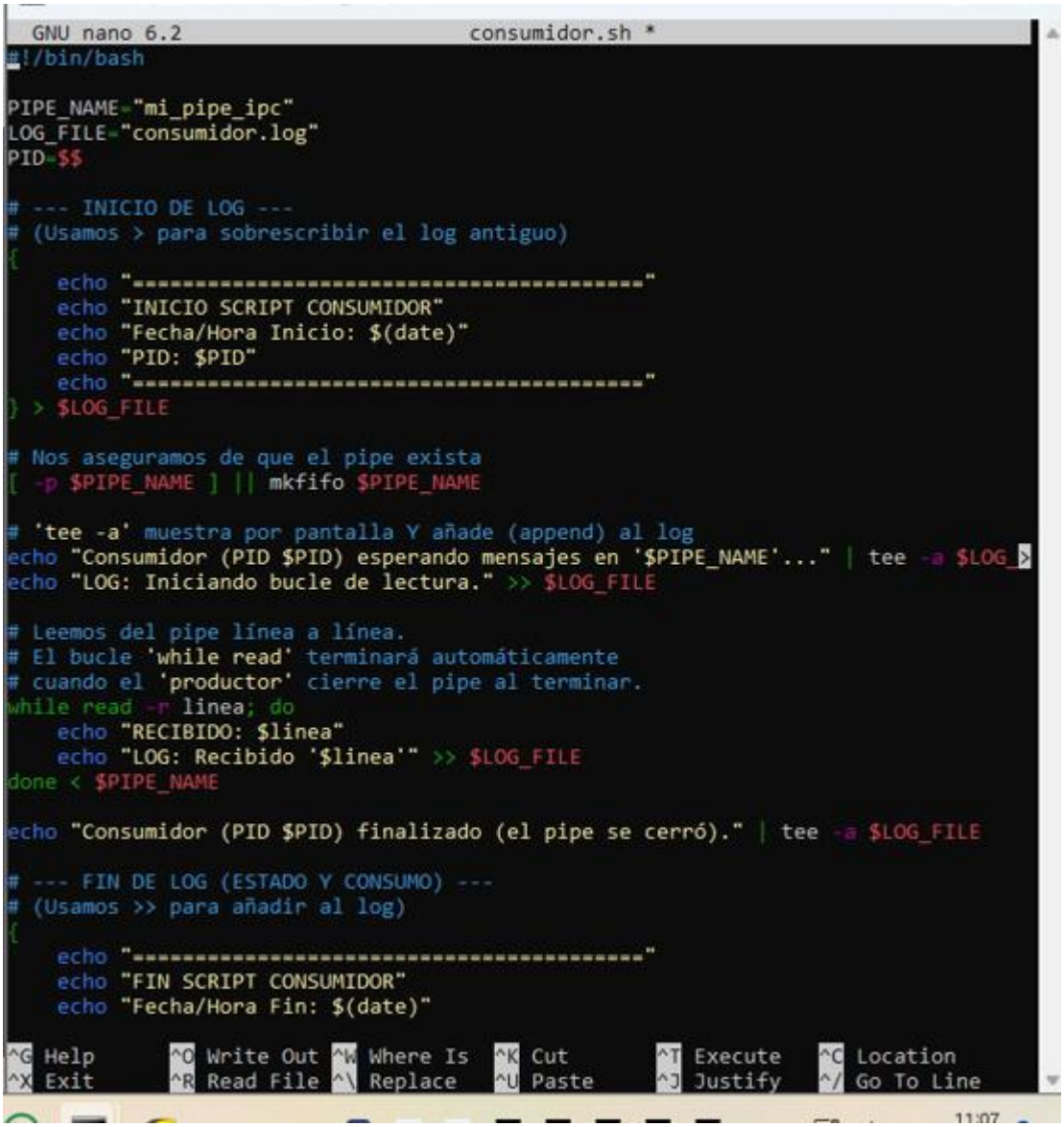
- *Fecha y hora de inicio.*
- *PID de cada proceso.*
- *Estado y consumo al finalizar.*

productor.sh

```
#!/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):"
```

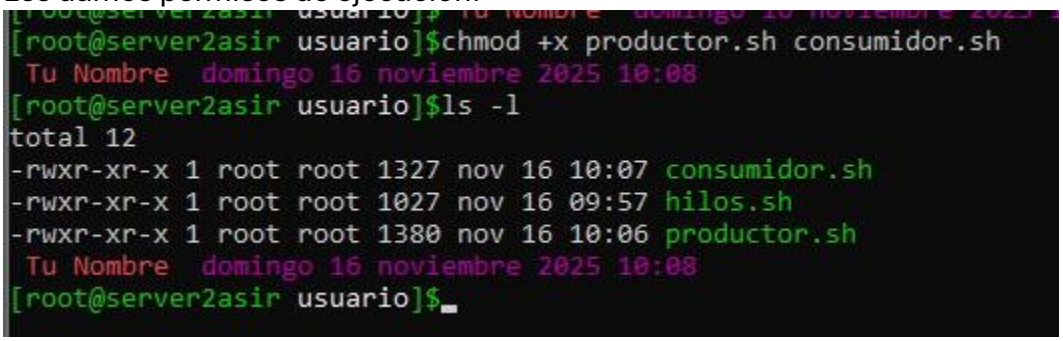
```
    # ps nos da una "foto" del consumo del proceso
    ps -p $PID -o pid,ppid,%cpu,%mem,etime,cmd
    echo "========================================"
} >> $LOG_FILE
```

consumidor.sh

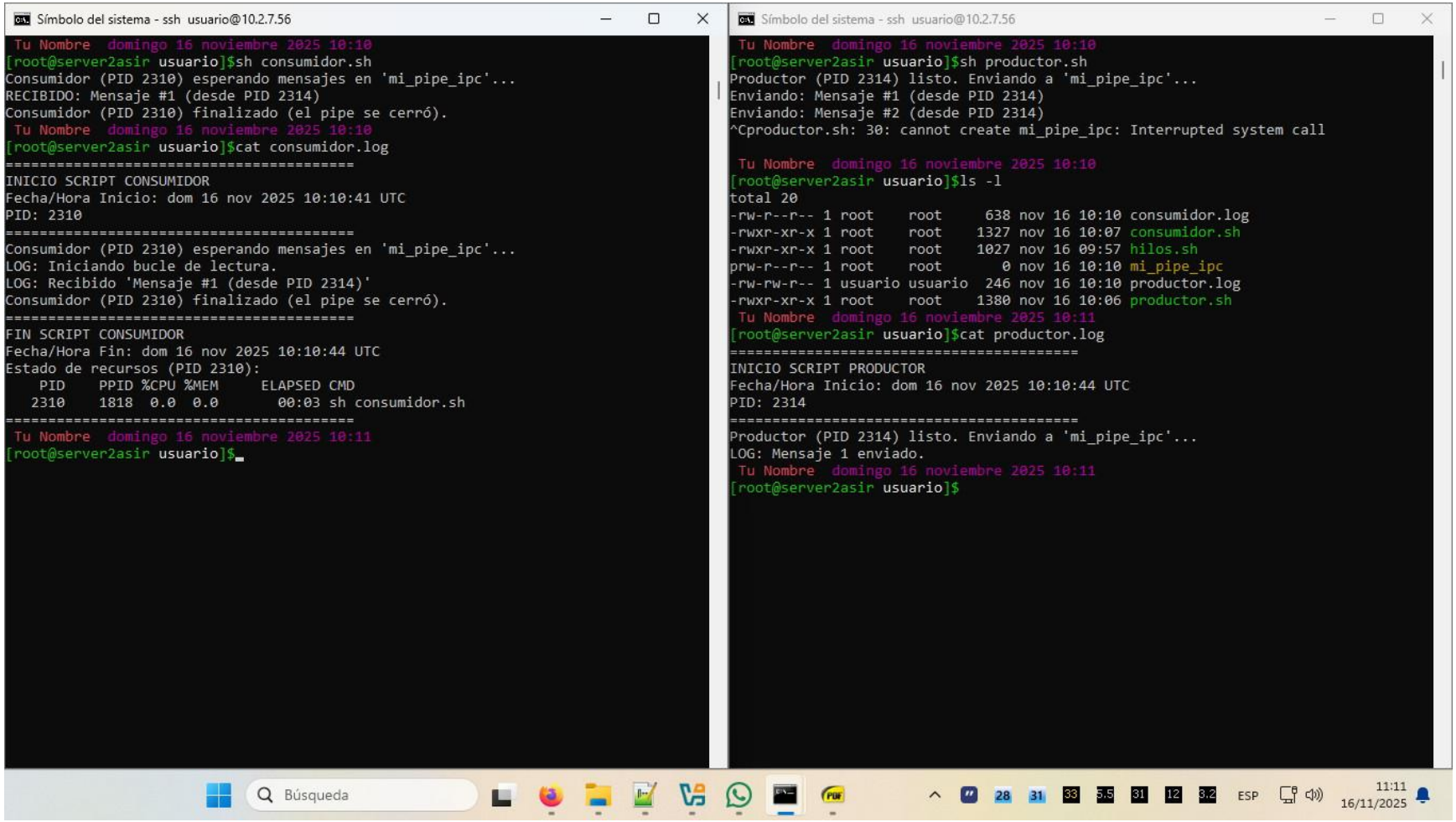


```
#!/bin/bash

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

# --- INICIO DE LOG ---
# (Usamos > para sobrescribir el log antiguo)
{
  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

# 'tee -a' muestra por pantalla Y añade (append) al log
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) ---
# (Usamos >> para añadir al log)
{
  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
```

Les damos permisos de ejecución:



Ahora debemos ejecutar primero el de consumidor.sh y después el de productor.sh
Para ello usaremos dos terminales a la vez. Abre otra sesión de SSH en mi caso.
Una vez realizado el envío comprueba que se han creado los logs de cada uno de ellos.
Revísalos.


Símbolo del sistema - ssh usuario@10.2.7.56
Tu Nombre domingo 16 noviembre 2025 10:10
[root@server2asir usuario]$sh consumidor.sh
Consumidor (PID 2310) esperando mensajes en 'mi_pipe_ipc'...
RECIBIDO: Mensaje #1 (desde PID 2314)
Consumidor (PID 2310) finalizado (el pipe se cerró).
Tu Nombre domingo 16 noviembre 2025 10:10
[root@server2asir usuario]$cat consumidor.log
==
INICIO SCRIPT CONSUMIDOR
Fecha/Hora Inicio: dom 16 nov 2025 10:10:41 UTC
PID: 2310
==
Consumidor (PID 2310) esperando mensajes en 'mi_pipe_ipc'...
LOG: Iniciando bucle de lectura.
LOG: Recibido 'Mensaje #1 (desde PID 2314)'
Consumidor (PID 2310) finalizado (el pipe se cerró).
==
FIN SCRIPT CONSUMIDOR
Fecha/Hora Fin: dom 16 nov 2025 10:10:44 UTC
Estado de recursos (PID 2310):
PID PPID %CPU %MEM ELAPSED CMD
2310 1818 0.0 0.0 00:03 sh consumidor.sh
==
Tu Nombre domingo 16 noviembre 2025 10:11
[root@server2asir usuario]$
Símbolo del sistema - ssh usuario@10.2.7.56
Tu Nombre domingo 16 noviembre 2025 10:10
[root@server2asir usuario]$sh productor.sh
Productor (PID 2314) listo. Enviando a 'mi_pipe_ipc'...
Enviando: Mensaje #1 (desde PID 2314)
Enviando: Mensaje #2 (desde PID 2314)
^Cproductor.sh: 30: cannot create mi_pipe_ipc: Interrupted system call
Tu Nombre domingo 16 noviembre 2025 10:10
[root@server2asir usuario]$ls -l
total 20
-rw-r--r-- 1 root root 638 nov 16 10:10 consumidor.log
-rwxr-xr-x 1 root root 1327 nov 16 10:07 consumidor.sh
-rwxr-xr-x 1 root root 1027 nov 16 09:57 hilos.sh
prw-r--r-- 1 root root 0 nov 16 10:10 mi_pipe_ipc
-rw-rw-r-- 1 usuario usuario 246 nov 16 10:10 productor.log
-rwxr-xr-x 1 root root 1380 nov 16 10:06 productor.sh
Tu Nombre domingo 16 noviembre 2025 10:11
[root@server2asir usuario]$cat productor.log
==
INICIO SCRIPT PRODUCTOR
Fecha/Hora Inicio: dom 16 nov 2025 10:10:44 UTC
PID: 2314
==
Productor (PID 2314) listo. Enviando a 'mi_pipe_ipc'...
LOG: Mensaje 1 enviado.
Tu Nombre domingo 16 noviembre 2025 10:11
[root@server2asir usuario]$
Búsqueda
ESP
11:11
16/11/2025

Analiza los logs y explica cómo pueden utilizarse para diagnosticar incidencias.

El objetivo principal de estos logs es establecer una **línea de tiempo** y correlacionar la actividad entre procesos independientes.

- **Fecha y Hora de Inicio/Fin:** Permite calcular el tiempo de ejecución y correlacionar eventos entre el Productor y el Consumidor. Si el inicio del Consumidor es mucho más tarde que el del Productor, puede haber una latencia en el arranque.
- **PID del Proceso:** Identifica unívocamente el proceso responsable de la acción. Útil para verificar si el proceso se "murió" o fue reemplazado por otro (PID diferente) en un reinicio inesperado.
- **Registros de Mensajes:** Confirma que los datos se transmiten correctamente, en el orden esperado y sin corrupción.
- **Estado y Consumo al Finalizar:** Proporciona una instantánea del estado final. Aunque en bash es simulado, en un log real (ej. top o ps) mostraría uso de CPU, Memoria o Estado de Salida (exit code).