

Experimento 3

GÓMEZ EDGAR, Universidad Central de Venezuela, Venezuela

Este experimento implementa la comunicación entre procesos utilizando tuberías (pipes) en Python. Se demuestra cómo un proceso padre puede crear múltiples hijos, recibir datos de ellos a través de `stdout`, y sincronizar su ejecución. Se exploran mecanismos de comunicación interprocesos (IPC) mediante `subprocess.PIPE` y `sys.stdout.flush()`, conceptos esenciales para sistemas distribuidos donde el intercambio de datos entre procesos es fundamental.

CCS Concepts: • **Organización de sistemas informáticos** → **Sistemas Distribuidos**; *Hilos en software*; *Procesos en software*.

Additional Key Words and Phrases: Sistemas Distribuidos, Arquitectura de Software, procesos en software, hilos en software

ACM Reference Format:

GÓMEZ EDGAR. 2025. Experimento 3. *Proc. ACM Hum.-Comput. Interact.* 1, 1, Article 1 (December 2025), 3 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1. Introducción

La comunicación entre procesos (IPC, por sus siglas en inglés) es un componente crítico en sistemas distribuidos y aplicaciones concurrentes. Mientras que experimentos anteriores se centraron en la creación y sincronización de procesos, este trabajo avanza hacia la transferencia de datos entre procesos padre e hijos utilizando tuberías (pipes). Se implementa un mecanismo donde múltiples hijos envían mensajes estructurados al padre a través de `stdout`, y el padre los captura mediante `subprocess.PIPE`. Este enfoque simula escenarios reales donde los procesos secundarios deben reportar resultados o estados a un proceso coordinador, una necesidad común en arquitecturas de software distribuidas y sistemas paralelos.

2. Proceso Padre

Se ha modificado el proceso padre para que pueda recibir datos de los hijos a través de los pipes `stdout` y `stdin`. El código espera la finalización de los procesos hijos con `wait()`, luego comprueba si el proceso hijo envió información al padre y, en caso afirmativo, la muestra en la consola.

```
import os
import subprocess

print(f"Hola, soy el proceso padre. Mi PID es: {os.getpid()}")
print("Creando procesos hijos...")

array = []
for i in range(5):
    # Iniciar el proceso hijo con stdout por pipe (el hijo enviará datos al padre)
    nuevo = subprocess.Popen(
        ["python", "hijo.py"],
        stdout=subprocess.PIPE, # Redirigir stdout del hijo a una tubería
        stderr=subprocess.PIPE,
        text=True, # Trabajar con texto en lugar de bytes
        bufsize=1, # Buffer en línea (line-buffered)
```

Author's Contact Information: GÓMEZ EDGAR, Universidad Central de Venezuela, Caracas, Venezuela.

2025. ACM 2573-0142/2025/12-ART1

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

```

        universal_newlines=True # Compatibilidad con nuevas líneas
    )
    array.append(nuevo)

for p in array:
    print(f"El proceso hijo se ha creado con el PID: {p.pid}")

# Esperar a cada hijo y leer lo que envió por stdout
for p in array:
    p.wait() # espera a que el hijo termine

    if p.stdout:
        salida = p.stdout.read() # leer todo lo enviado por el hijo
        p.stdout.close()
    else:
        salida = ""

    print(f"Salida del hijo PID {p.pid}: \n{salida.strip()}")
    print(f"El proceso hijo con PID {p.pid} ha terminado.")

print("Adiós, soy el proceso padre.")

```

3. Proceso Hijo

El proceso hijo envía información al padre escribiendo usando `sys.stdout.write()` y luego enviando los datos con `sys.stdout.flush()`. También tiene una espera de cinco segundos simulando una carga de trabajo realizada por el proceso hijo.

```

import os
import time
import sys

sys.stdout.write(f"Hola, soy el proceso hijo. Mi PID es: {os.getpid()}. \n")
sys.stdout.flush()
time.sleep(5) # Simula una tarea que toma 5 segundos
sys.stdout.write("El proceso hijo ha terminado su tarea. \n")
sys.stdout.flush()

```

4. Resultados de la ejecución

Ejecutando el código mostrado de los procesos obtenemos en la terminal la siguiente respuesta, como se observa en la Fig 1.

```

● PS C:\Users\Edgar Gomez\Desktop\Tareas\Tarea3.1> python padre.py
Hola, soy el proceso padre. Mi PID es: 20968
Creando procesos hijos...
El proceso hijo se ha creado con el PID: 19860
El proceso hijo se ha creado con el PID: 8068
El proceso hijo se ha creado con el PID: 24768
El proceso hijo se ha creado con el PID: 24584
El proceso hijo se ha creado con el PID: 24744
Salida del hijo PID 19860:
Hola, soy el proceso hijo. Mi PID es: 19860.
El proceso hijo ha terminado su tarea.
● El proceso hijo con PID 19860 ha terminado.
Salida del hijo PID 8068:
Hola, soy el proceso hijo. Mi PID es: 8068.
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 8068 ha terminado.
Salida del hijo PID 24768:
Hola, soy el proceso hijo. Mi PID es: 24768.
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 24768 ha terminado.
Salida del hijo PID 24584:
Hola, soy el proceso hijo. Mi PID es: 24584.
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 24584 ha terminado.
Salida del hijo PID 24744:
Hola, soy el proceso hijo. Mi PID es: 24744.
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 24744 ha terminado.
Adiós, soy el proceso padre.

```

Fig. 1. Resultado de los hijos enviando información al padre.

Se pueden observar los PID de cada proceso, lo que demuestra que son procesos completamente diferentes. Se muestra la creación de cinco procesos hijos con PID únicos. Se logra apreciar los datos enviados desde los procesos hijos al padre luego de la línea que señala “Salida del hijo PID: X”, con los mensajes: “Hola soy el proceso hijo. Mi PID es: X” y “El proceso hijo ha terminado su tarea”. Estos dos mensajes fueron enviados desde el hijo hasta el padre y el padre fue el encargado de mostrarlos en consola.

5. Conclusión

El experimento demostró exitosamente la comunicación unidireccional de hijo a padre mediante tuberías (pipes) en Python. Se verificó que el proceso padre puede crear múltiples hijos, recibir datos de ellos a través de stdout, y sincronizar su ejecución utilizando `wait()`. Los mecanismos de comunicación interprocesos (IPC) implementados con `subprocess.PIPE` y `sys.stdout.flush()` permitieron un intercambio de datos efectivo, evidenciando su importancia en sistemas distribuidos donde la transferencia de resultados desde procesos hijos a un proceso coordinador es fundamental. Este patrón es aplicable en arquitecturas maestro-trabajador y sistemas de procesamiento paralelo.