Experimento 5

GÓMEZ EDGAR, Universidad Central de Venezuela, Venezuela

Este experimento agrego la comunicación del padre a los hijos mediante argumentos de línea de comandos en Python. Se demuestra cómo un proceso padre puede enviar parámetros personalizados a múltiples hijos durante su creación. Este mecanismo permite al padre configurar individualmente el comportamiento de cada proceso hijo, estableciendo un patrón de comunicación inicial esencial en arquitecturas maestro-trabajador de sistemas distribuidos.

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 5. *Proc. ACM Hum.-Comput. Interact.* 1, 1, Article 1 (December 2025), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1. Introducción

Este experimento se enfoca en la comunicación del padre hacia los hijos mediante argumentos de línea de comandos. A diferencia de enfoques previos que utilizaban tuberías para comunicación durante la ejecución, aquí el padre envía información a los hijos en el momento mismo de su creación. Se implementa un sistema donde cada hijo recibe dos parámetros específicos: un mensaje identificador único y un tiempo de espera personalizado que determina la duración de su tarea simulada. Este mecanismo permite inicializar procesos hijos con configuraciones diferenciadas, simulando escenarios reales donde un coordinador debe asignar trabajos con parámetros específicos a múltiples trabajadores en sistemas distribuidos.

2. Proceso Padre

Hemos modificado el código del padre para que pase argumentos al proceso hijo como una fomar de comunicación del proceso padre a sus hijos. Pasamos como argumento un mensaje y como segundo argumento el tiempo de espera del hijo que simula una carga de trabajo en el hijo.

```
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", f"Mensaje_desde_padre_{i}", str(i * 2)],
   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

Se ha modificado el proceso hijo para que reciba dos argumentos, el primero un mensaje del padre y el segundo el tiempo de espera que simula una carga de trabajo.

```
 import os
 import time
 import sys

mensaje_desde_el_padre = sys.argv[1] if len(sys.argv) > 1 else "No se recibió ningún mensaje del padre."
 sleep_desde_el_padre = int(sys.argv[2]) if len(sys.argv) > 2 else 1

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

4. Resultados de la ejecución

```
PS C:\Users\Edgar Gomez\Desktop\Tareas\Tarea3.1> python padre.py
Hola, soy el proceso padre. Mi PID es: 38936
Creando procesos hijos...
El proceso hijo se ha creado con el PID: 15764
El proceso hijo se ha creado con el PID: 37340
El proceso hijo se ha creado con el PID: 28796
El proceso hijo se ha creado con el PID: 34512
El proceso hijo se ha creado con el PID: 32568
Salida del hijo PID 15764:
Hola, soy el proceso hijo. Mi PID es: 15764.
Mensaje recibido del padre: Mensaje_desde_padre_0
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 15764 ha terminado.
Salida del hijo PID 37340:
Hola, soy el proceso hijo. Mi PID es: 37340.
Mensaje recibido del padre: Mensaje_desde_padre_1
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 37340 ha terminado.
Salida del hijo PID 28796:
Hola, soy el proceso hijo. Mi PID es: 28796.
Mensaje recibido del padre: Mensaje_desde_padre_2
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 28796 ha terminado.
Salida del hijo PID 34512:
Hola, soy el proceso hijo. Mi PID es: 34512.
Mensaje recibido del padre: Mensaje_desde_padre_3
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 34512 ha terminado.
Salida del hijo PID 32568:
Hola, soy el proceso hijo. Mi PID es: 32568.
Mensaje recibido del padre: Mensaje_desde_padre_4
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 32568 ha terminado.
Adiós, soy el proceso padre.
```

Fig. 1. Comunicación desde el padre al hijo.

La ejecución demuestra una comunicación efectiva del padre hacia los hijos mediante argumentos de línea de comandos. En el momento de crear cada proceso hijo, el padre envía dos parámetros específicos: un mensaje personalizado (`Mensaje_desde_padre_X`) y un tiempo de espera diferenciado (`i * 2` segundos). Cada hijo recibe estos argumentos a través de `sys.argv` y los utiliza directamente en su ejecución, mostrando el mensaje recibido y ajustando su tiempo de simulación de trabajo según el valor especificado por el padre.

5. Conclusión

El experimento demostró exitosamente la comunicación del padre hacia los hijos mediante argumentos de línea de comandos. Se verificó que el padre puede enviar parámetros personalizados (mensajes y tiempos de espera diferenciados) a cada hijo durante su creación, permitiendo configurar individualmente su comportamiento antes de que inicien su ejecución. Este mecanismo de comunicación inicial es especialmente útil en arquitecturas maestro-trabajador donde se requiere asignar tareas con configuraciones específicas desde el inicio. El enfoque implementado proporciona un método eficiente para la parametrización de procesos hijos en sistemas distribuidos, estableciendo las bases para una ejecución controlada y configurable de trabajos concurrentes.