Experimento 2

GÓMEZ EDGAR, Universidad Central de Venezuela, Venezuela

Este experimento implementa la creación y gestión de múltiples procesos hijos desde un proceso padre en Python. Se demuestra la capacidad de generar y sincronizar cinco procesos concurrentes utilizando `subprocess.Popen` y `wait()`. Se analiza el comportamiento de concurrencia, la asignación de PID únicos y la coordinación entre procesos, conceptos esenciales para el desarrollo de sistemas distribuidos y aplicaciones paralelas.

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

1. Introducción

La gestión eficiente de múltiples procesos concurrentes es fundamental en sistemas distribuidos y aplicaciones modernas que requieren paralelismo. Este experimento explora la creación y coordinación de varios procesos hijos desde un único proceso padre en Python. A diferencia del modelo padre-hijo simple, este enfoque permite analizar patrones de concurrencia real, donde múltiples tareas se ejecutan simultáneamente mientras se mantiene la capacidad de sincronización. Se examinan conceptos como la asignación de PID, la ejecución no determinista en entornos concurrentes y la importancia de mecanismos de espera para garantizar la finalización ordenada de procesos.

2. Proceso Padre

El proceso padre realiza los siguientes pasos secuencialmente, muestra el PID del proceso padre, luego se inician cinco procesos hijos, posteriormente se muestran los PID de los porcesos hijos, se espera que termine los preoceos hijos y luego se muestra un mensaje de que ha finalizado el proceso padre.

```
import os
import subprocess

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

# Iniciar el proceso hijo
# 'subprocess.Popen' permite que el padre continúe su ejecución
array = []
for i in range(5):
 new_proceso_hijo = subprocess.Popen(["python", "hijo.py"])
 array.append(new_proceso_hijo)

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

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

```
# El proceso padre espera a que el hijo termine
# Esto demuestra la relación padre-hijo
for proceso_hijo in array:
 proceso_hijo.wait()
 print(f"El proceso hijo con PID {proceso_hijo.pid} ha terminado.")

print("Adiós, soy el proceso padre.")
```

3. Proceso Hijo

El procesos hijo muestra su PID, espera cinco segundos y luego muestra un mensaje de que finalizado la tarea.

```
import os
import time

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

4. Resultados de la ejecución

Ejecuanto 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: 3016
Creando procesos hijos...
El proceso hijo se ha creado con el PID: 20880
El proceso hijo se ha creado con el PID: 23100
El proceso hijo se ha creado con el PID: 14200
El proceso hijo se ha creado con el PID: 11516
El proceso hijo se ha creado con el PID: 21788
Hola, soy el proceso hijo. Mi PID es: 20880
Hola, soy el proceso hijo. Mi PID es: 23100
Hola, soy el proceso hijo. Mi PID es: 14200
Hola, soy el proceso hijo. Mi PID es: 11516
Hola, soy el proceso hijo. Mi PID es: 21788
El proceso hijo ha terminado su tarea.
El proceso hijo ha terminado su tarea.
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 20880 ha terminado.
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 23100 ha terminado.
El proceso hijo ha terminado su tarea.
El proceso hijo con PID 14200 ha terminado.
El proceso hijo con PID 11516 ha terminado.
El proceso hijo con PID 21788 ha terminado.
Adiós, soy el proceso padre.
```

Fig. 1. resultado de ejecutar multiples procesos hijos

Se pueden observar los PID de cada proceso que de muestra que son procesos completamente diferentes, se muestra la creacion de cinco procesos hijos con PID únicos, tambien se puede observar que la ejecución es concurente y que no existe un orden en que los mensajes son mostrados, se observa que al final el proceso padre espera la terminación de todos los procesos hijos.

5. Conclusión

El experimento demostró exitosamente la creación y gestión de múltiples procesos hijos concurrentes. Se observó que cada proceso recibe un PID único y que la ejecución presenta características no deterministas, donde el orden de los mensajes varía entre ejecuciones. El uso de `wait()` permitió al padre sincronizar la finalización de todos los hijos, asegurando una terminación ordenada. Estos resultados destacan la importancia de los mecanismos de coordinación en aplicaciones paralelas y distribuidas, donde la gestión adecuada de múltiples procesos es crucial para el rendimiento y la confiabilidad del sistema.