Experimento 1

GÓMEZ EDGAR, Universidad Central de Venezuela, Venezuela

Este experimento implementa la creación y gestión de procesos en Python, demostrando la relación padre-hijo entre procesos. Se comparan dos escenarios: ejecución secuencial donde el padre espera la finalización del hijo mediante `wait()`, y ejecución concurrente donde ambos procesos se ejecutan independientemente. Se analizan conceptos clave como PID, sincronización de procesos y la aparición de procesos huérfanos, relevantes para el diseño de sistemas distribuidos y aplicaciones concurrentes.

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

1. Introducción

En el ámbito de los sistemas operativos y la programación de sistemas, la gestión de procesos es un concepto fundamental que permite la ejecución concurrente de múltiples tareas. Los procesos pueden relacionarse entre sí a través de una jerarquía padre-hijo, donde un proceso principal (padre) puede crear y gestionar uno o más procesos secundarios (hijos). Este modelo de ejecución es esencial en sistemas distribuidos y aplicaciones multiproceso, ya que permite la división de trabajo, el aislamiento de tareas y una mejor utilización de los recursos del sistema.

El presente experimento tiene como objetivo demostrar la creación y gestión de procesos en Python, específicamente la relación entre un proceso padre y un proceso hijo. A través de la implementación de un programa sencillo, se exploran dos escenarios clave: la ejecución secuencial, donde el padre espera a que el hijo finalice su tarea, y la ejecución concurrente, donde ambos procesos se ejecutan de manera independiente. Estos casos permiten observar el comportamiento del sistema operativo ante diferentes estrategias de gestión de procesos, así como conceptos como los PID (Process ID), la espera activa y los procesos huérfanos.

La relevancia de este experimento radica en su aplicabilidad a sistemas distribuidos y arquitecturas de software modernas, donde la concurrencia y la paralelización son requisitos comunes para mejorar el rendimiento y la escalabilidad de las aplicaciones.

2. Proceso Padre

El proceso padre realiza los siguientes pasos secuencialmente, muestra el PID del proceso padre, luego se inicia un proceso hijo posteriormente se muestra el PID del porceso hijo, se espera que termine el proceso hijo y luego mostramos un mensaje de que el proceso hijo a terminado y luego que el proceso a finalizado.

```
import os
import subprocess

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

# Iniciar el proceso hijo
# 'subprocess.Popen' permite que el padre continúe su ejecución
```

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

```
proceso_hijo = subprocess.Popen(["python", "hijo.py"])

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

# El proceso padre espera a que el hijo termine
# Esto demuestra la relación padre-hijo
proceso_hijo.wait()

print("El proceso hijo ha terminado. El proceso padre ha finalizado su espera.")
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 Fig 1.

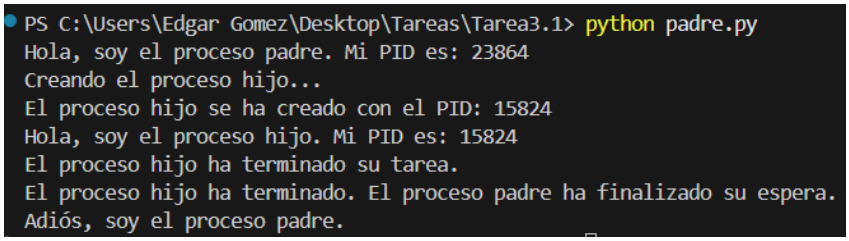

Fig. 1. En el padre se espera a que termine el proceso hijo.

Se pueden observar los PID de cada proceso que de muestra que son procesos completamente diferentes, en la ejecución se pudo notar que el proceso padre espero a que terminará la ejecución de su proceso hijo antes de que el proceos padre finalizara.

Para la ejecución concurrente, como se indica el Experimento 1, comentamos está línea de codigo del padre que espera a que el proceso hijo finalice. El resutlado de la ejecución se observa en Fig 2.

```
#proceso_hijo.wait()
```

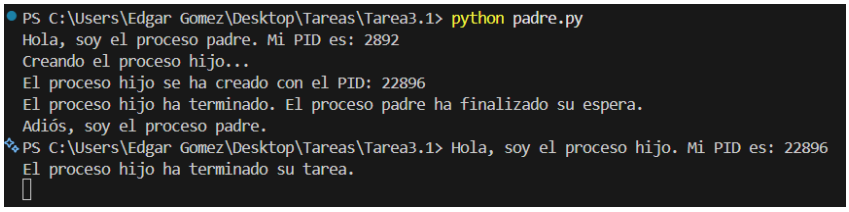

Fig. 2. En el padre no se espera a que termine el proceso hijo

Esto arroja en nuestara terminal que el proceso padre finalice, antes de esperar que finalice el proceso hijo, no espera los 5 segundos que tarda en finlizar el proceso hijo. Tambien se observa que el proceso hijo sigue ejecutandose incluso despúes que ha finalizado el padre, el hijo se convierte en un porceso huerfano.

5. Conclusión

A través de la implementación y ejecución del experimento, se pudo observar de manera práctica los principios fundamentales de la gestión de procesos en sistemas operativos. En primer lugar, se confirmó que cada proceso, ya sea padre o hijo, posee un identificador único (PID) que lo distingue en el sistema. Además, se demostró que la relación padre-hijo permite al proceso principal ejercer control sobre los procesos secundarios, particularmente mediante la instrucción `wait()`, que suspende la ejecución del padre hasta que el hijo finalice.

El segundo escenario, donde se omitió la espera del padre, evidenció el comportamiento concurrente de los procesos y la aparición de procesos huérfanos cuando el padre finaliza antes que el hijo. Este fenómeno es relevante en entornos de servidores y sistemas de larga duración, donde una gestión inadecuada de los procesos hijos puede generar fugas de recursos o comportamientos inesperados.

En resumen, el experimento no solo ilustró los mecanismos básicos de creación y sincronización de procesos, sino que también destacó la importancia de diseñar estrategias adecuadas de concurrencia según los requisitos de la aplicación. Estos conocimientos son esenciales para el desarrollo de software eficiente y confiable en sistemas distribuidos y entornos multihilo, donde la correcta coordinación entre procesos determina en gran medida el desempeño y la estabilidad del sistema.