**PRÁCTICA 3 DE SISTEMAS OPERATIVOS**

**TEMA: CREACIÓN DE PROCESOS**

**Nombre: Fernando Eliceo Huilca Villagómez**

**Carrera: Software**

**Grupo: GR1SW**

**Fecha: 09/07/2024**

**Índice de Contenidos:**

[1. OBJETIVOS 2](#_Toc171535296)

[1.1. Familiarizar al estudiante con el uso de las funciones fork, system, exec. 2](#_Toc171535297)

[1.2. Realizar varias actividades de creación de procesos padres e hijos. 2](#_Toc171535298)

[2. INFORME 3](#_Toc171535299)

[3.1 USO DE FORK 3](#_Toc171535300)

[3.1.1 Ejecute el programa en segundo plano 4](#_Toc171535301)

[3.1.2 Creación de procesos hijos 7](#_Toc171535302)

[3.1.4 Creación de un proceso hijo sin espera. 8](#_Toc171535303)

[3.1.5 Creación de un proceso hijo con espera. 8](#_Toc171535304)

[3.1.7 Ejecución concurrente de procesos 11](#_Toc171535305)

[3.2 USO DE EXEC 13](#_Toc171535306)

[3.2.1 Creación de un proceso zombie 15](#_Toc171535307)

[3.2.2 Uso de exec 17](#_Toc171535308)

[3.2.3 Crear un proceso hijo y colocarlo dentro de un bucle infinito. 18](#_Toc171535309)

[3. CONCLUSIONES Y RECOMENDACIONES 19](#_Toc171535310)

[4. BIBLIOGRAFÍA 19](#_Toc171535311)

**Índice de Figuras**

[Ilustración 1 Uso de fork para familiarizarse con el concepto 3](#_Toc171536208)

[Ilustración 2 Uso de getppid y pid en procesos 3](#_Toc171536209)

[Ilustración 3 Ejecución de un proceso padre y su hijo 4](#_Toc171536210)

[Ilustración 4 Código modificado para que imprima el mensaje de padre primero 5](#_Toc171536211)

[Ilustración 5 Código que imprime sumatorio propio del padre e hijo 6](#_Toc171536212)

[Ilustración 6 Imprimir 10 veces soy el hijo y soy el padre 7](#_Toc171536213)

[Ilustración 7 Proceso hijo sin espera. 8](#_Toc171536214)

[Ilustración 8 Proceso hijo con espera. 9](#_Toc171536215)

[Ilustración 9 Mensaje de finalización 10 segundos más tarde que el proceso hijo. 10](#_Toc171536216)

[Ilustración 10 Proceso padre y de tres procesos hijos 10](#_Toc171536217)

[Ilustración 11 Código que imprime número el de procesos hijos que se ejecutaron. 11](#_Toc171536218)

[Ilustración 12 Ejecución concurrente de procesos 12](#_Toc171536219)

[Ilustración 13 Código para que aparezca al final en pantalla el número de procesos hijos 13](#_Toc171536220)

[Ilustración 14 Uso de exec 14](#_Toc171536221)

[Ilustración 15 Se ha cambiado la función wait por sleep 15](#_Toc171536222)

[Ilustración 16 Creación de un proceso zombie 15](#_Toc171536223)

[Ilustración 17 Código que soluciona el problema planteado 16](#_Toc171536224)

[Ilustración 18 Proceso hijo que liste los procesos del sistema usando execl 17](#_Toc171536225)

[Ilustración 19 Creación de un proceso hijo colocado dentro de un bucle infinito. 18](#_Toc171536226)

# OBJETIVOS

## Familiarizar al estudiante con el uso de las funciones fork, system, exec.

## Realizar varias actividades de creación de procesos padres e hijos.

# INFORME

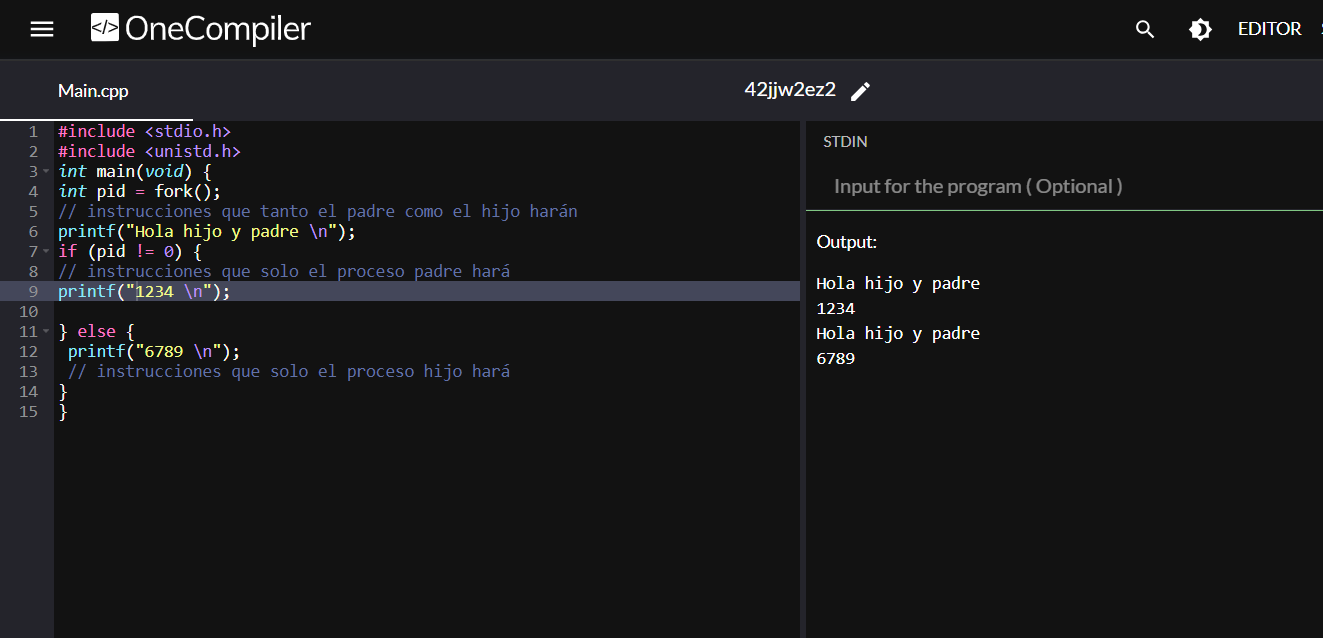
****

Ilustración 1 Uso de fork para familiarizarse con el concepto

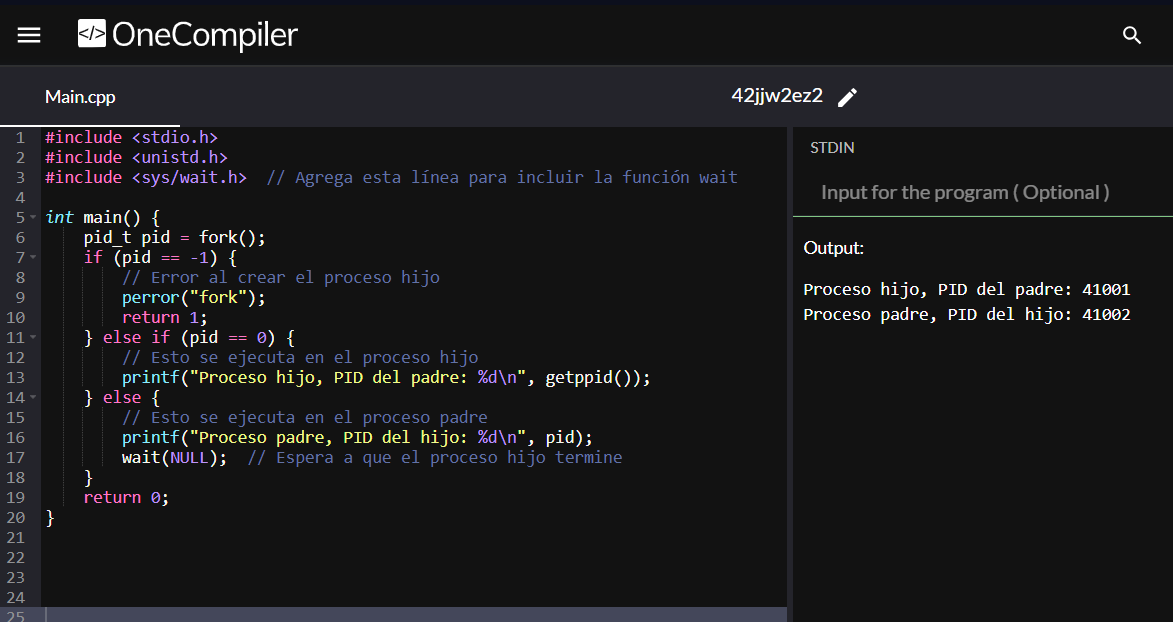
****

Ilustración 2 Uso de getppid y pid en procesos

**3. PROCEDIMIENTO**

## 3.1 USO DE FORK

Ejecutar cada uno de los códigos propuestos y contestar las preguntas:

## 3.1.1 Ejecute el programa en segundo plano

**Identifique los valores PID y PPID de cada proceso.**

El PID (Process ID) es un identificador único para cada proceso en ejecución. El PPID (Parent Process ID) es el identificador del proceso padre.

En el código:

* El PID del proceso padre se obtiene con getpid().
* El PID del proceso hijo se obtiene con getpid().
* El PPID del proceso hijo se obtiene con getppid().

Cuando se ejecutó el programa, se observó esto:

* PID del proceso padre: 40516
* PID del proceso hijo: 40517
* PPID del proceso hijo: 40518

El proceso padre tiene un PID de 40516 y un PPID de 40516, mientras que el proceso hijo tiene un PID de 40517 y un PPID de 40516.

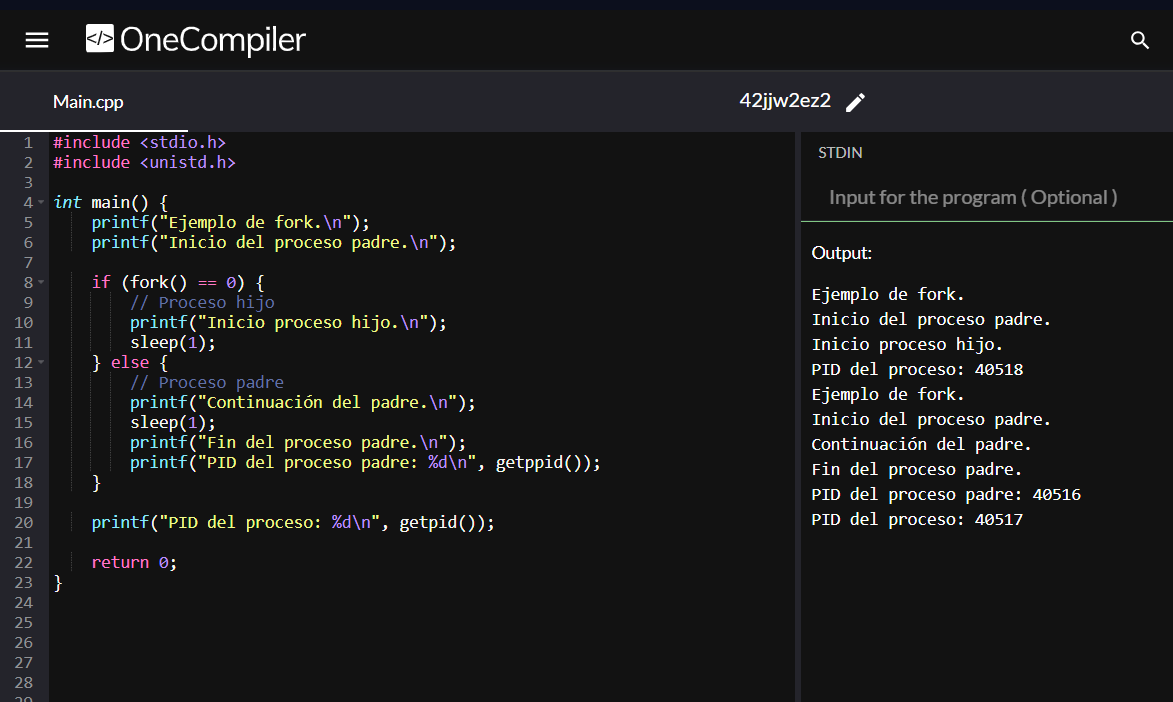


Ilustración 3 Ejecución de un proceso padre y su hijo

**¿Qué realiza la función sleep? ¿Qué proceso concluye antes de su ejecución?**

1. La función sleep(1) detiene la ejecución del proceso que la llama durante 1 segundo. En el caso del código, tanto el proceso hijo como el proceso padre llaman a sleep(1). No se puede determinar de antemano cuál de los procesos concluye primero, ya que ambos se ejecutan en paralelo y su finalización depende del sistema operativo y de la carga de trabajo en el momento.

**¿Qué ocurre cuando la llamada al sistema fork devuelve un valor negativo?**

1. Cuando fork() devuelve un valor negativo, indica que la creación del proceso hijo ha fallado. Esto puede ocurrir debido a la falta de recursos del sistema, como memoria insuficiente o límites de procesos alcanzados. En tal caso, fork() retorna -1 y típicamente se utiliza perror("fork") para imprimir un mensaje de error.

**¿Cuál es la primera instrucción que ejecuta el proceso hijo?**

1. La primera instrucción que ejecuta el proceso hijo en el código proporcionado es:
2. printf("Inicio proceso hijo.\n");

**Modifique el código para que el proceso padre imprima su mensaje antes que el hijo.**

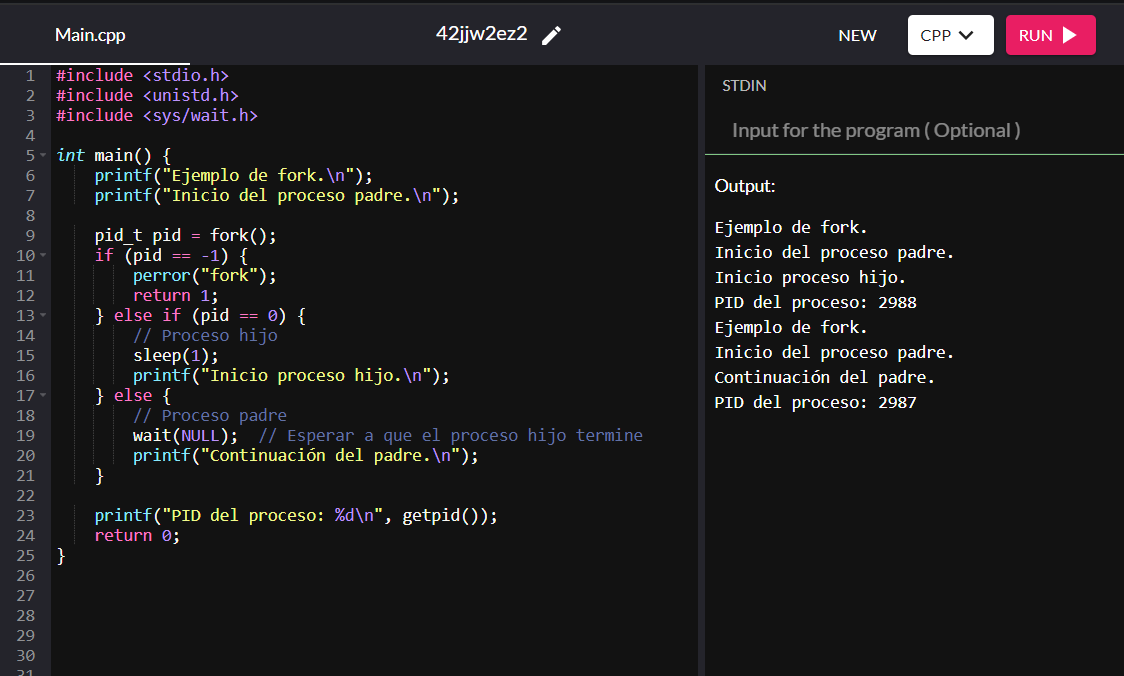
1. 

Ilustración 4 Código modificado para que imprima el mensaje de padre primero

**Modifique el código fuente del programa declarando una variable entera llamada proc e inicializándola a 10. Dicha variable deberá incrementarse 10 veces en el padre y de 10 en 10. Mientras que el hijo la incrementará 10 veces de 1 en 1. ¿Cuál será el valor final de la variable para el padre y para el hijo?**

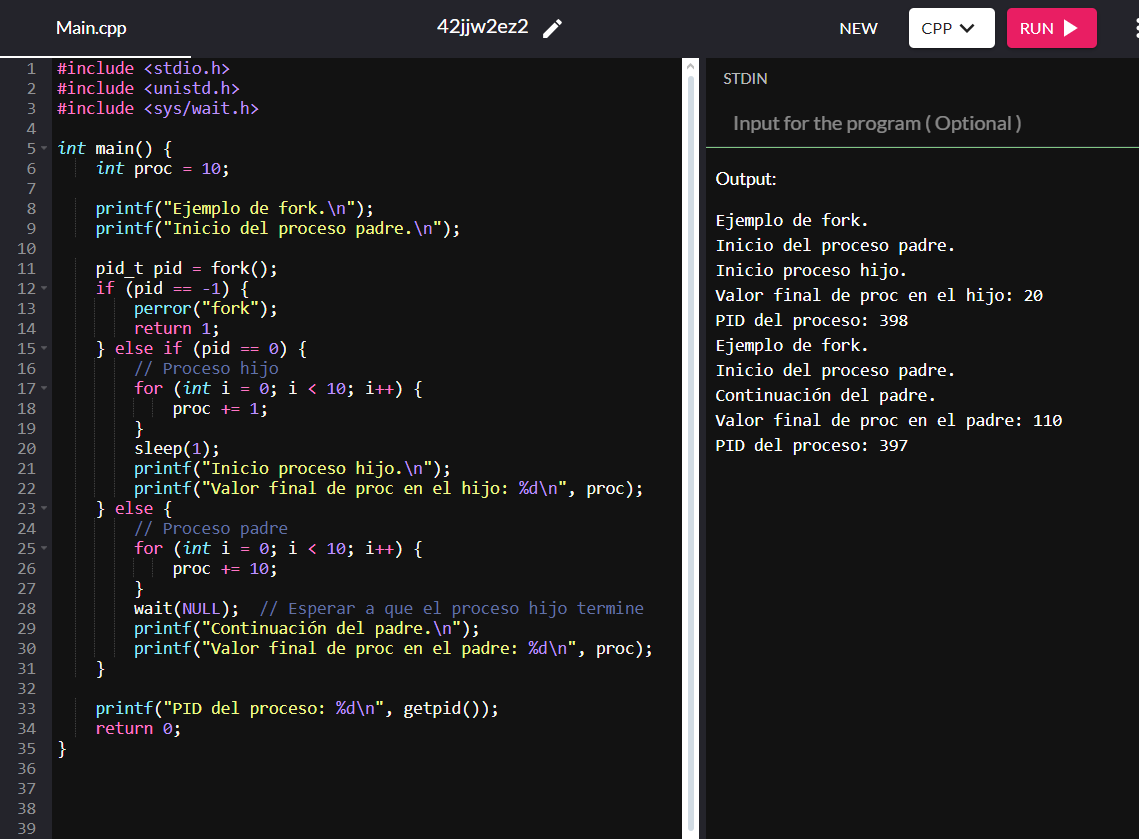


Ilustración 5 Código que imprime sumatorio propio del padre e hijo

## 3.1.2 Creación de procesos hijos

1. **Creación de un proceso padre que escribe 10 veces “Soy el padre” y de un proceso hijo que escribe 10 veces “Soy el hijo”.**

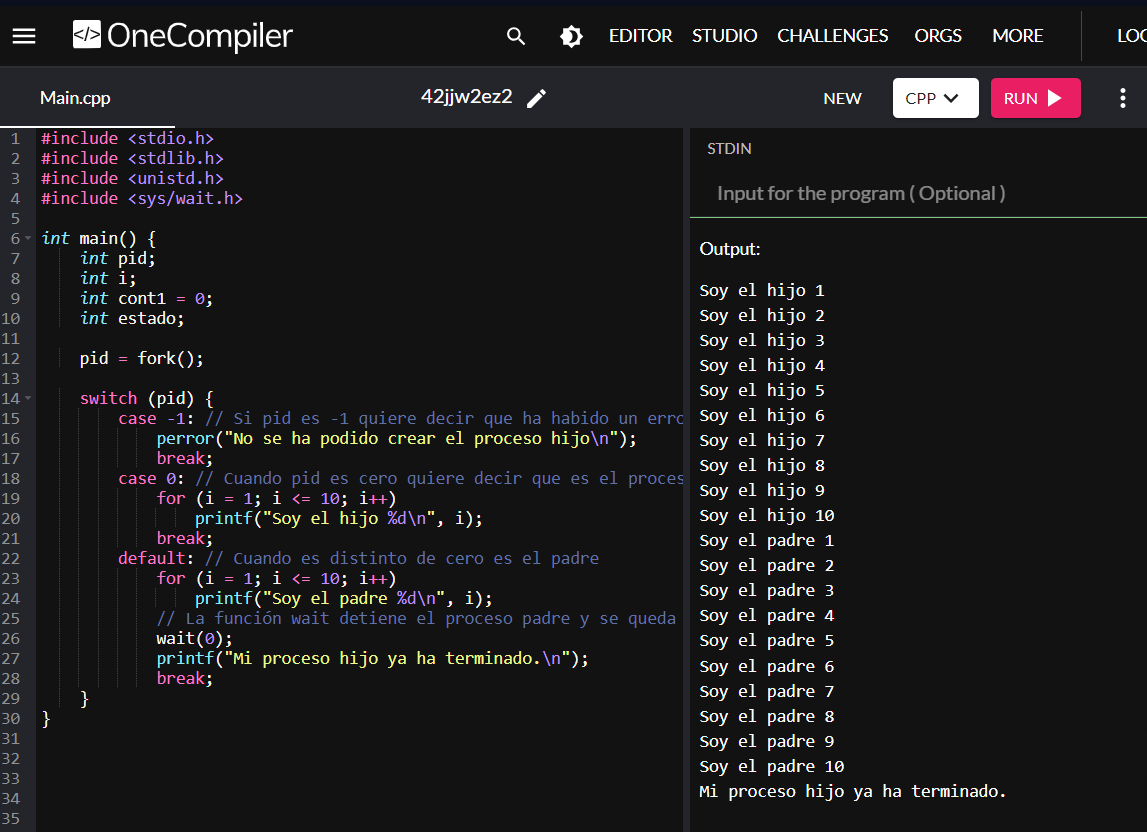


Ilustración 6 Imprimir 10 veces soy el hijo y soy el padre

**¿Qué realiza la función wait?**La función wait en C es utilizada por un proceso para esperar a que uno de sus procesos hijos termine su ejecución.

1. **Esperar la finalización del proceso hijo**: La función wait hace que el proceso padre se detenga y espere hasta que uno de sus procesos hijos termine su ejecución. Esto evita que el proceso padre continúe ejecutándose hasta que el proceso hijo haya terminado.
2. **Recibir el estado de salida del proceso hijo**: wait también permite que el proceso padre reciba el estado de salida del proceso hijo, lo cual puede incluir información sobre si el proceso hijo terminó normalmente o si fue interrumpido por una señal.
3. **Liberar recursos**: Cuando un proceso hijo termina, su entrada en la tabla de procesos aún consume recursos del sistema hasta que el proceso padre llama a wait. Al llamar a wait, el proceso padre libera estos recursos, lo cual es importante para evitar la acumulación de procesos "zombies".

## 3.1.4 Creación de un proceso hijo sin espera.

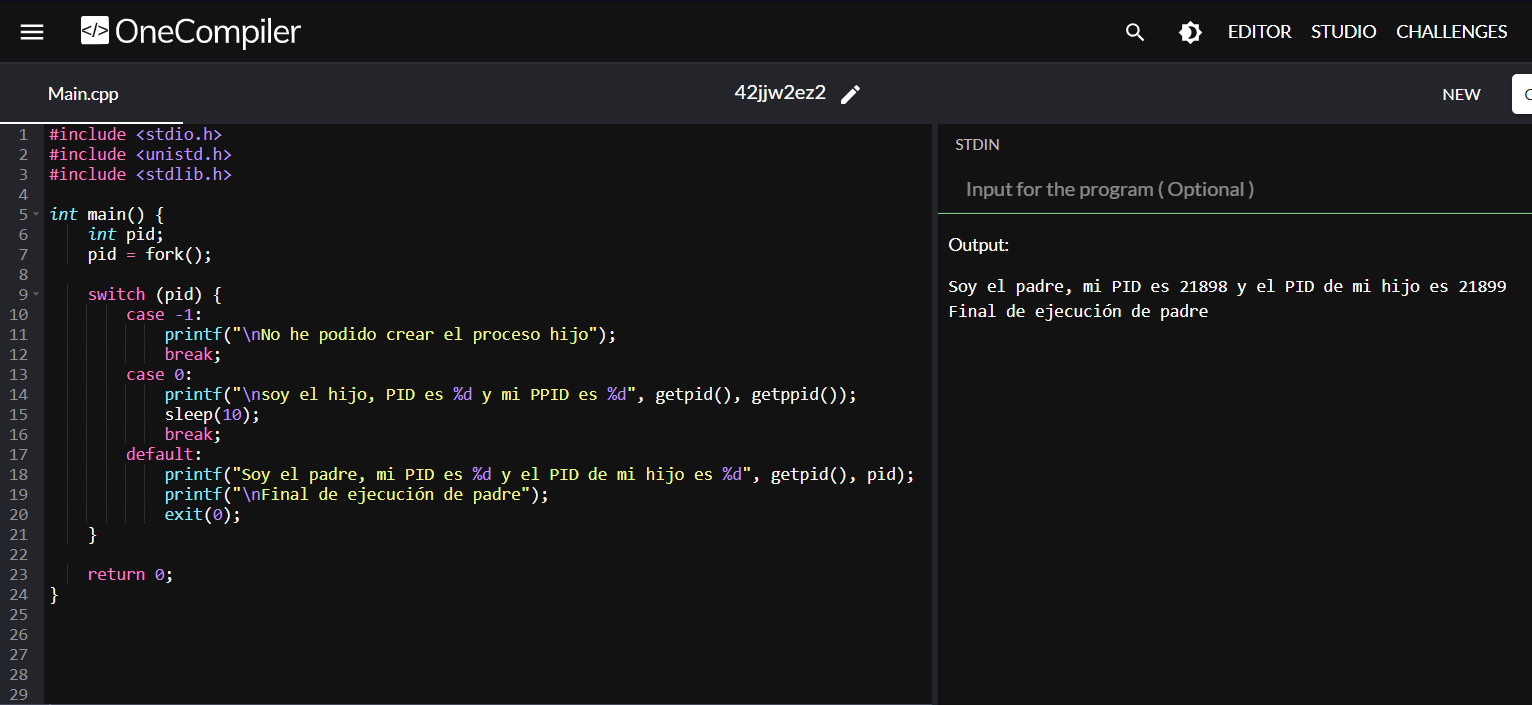
1. Ejecutar en segundo plano.
2. 

Ilustración 7 Proceso hijo sin espera.

1. **¿Cuál es el PPID del proceso hijo?**
2. El PPID (Parent Process ID) del proceso hijo es 21898. Esto se puede ver en la salida del programa, donde el proceso hijo imprime "Soy el hijo, mi PID es 21899 y el PID de mi padre es 21898".
3. **¿Cómo se denomina al tipo de proceso hijo?**
4. El tipo de proceso hijo se denomina "proceso fork". Esto se debe a que el programa utiliza la función fork() para crear un nuevo proceso hijo a partir del proceso padre.

## 

## Creación de un proceso hijo con espera.

**Ejecutar en segundo plano.**

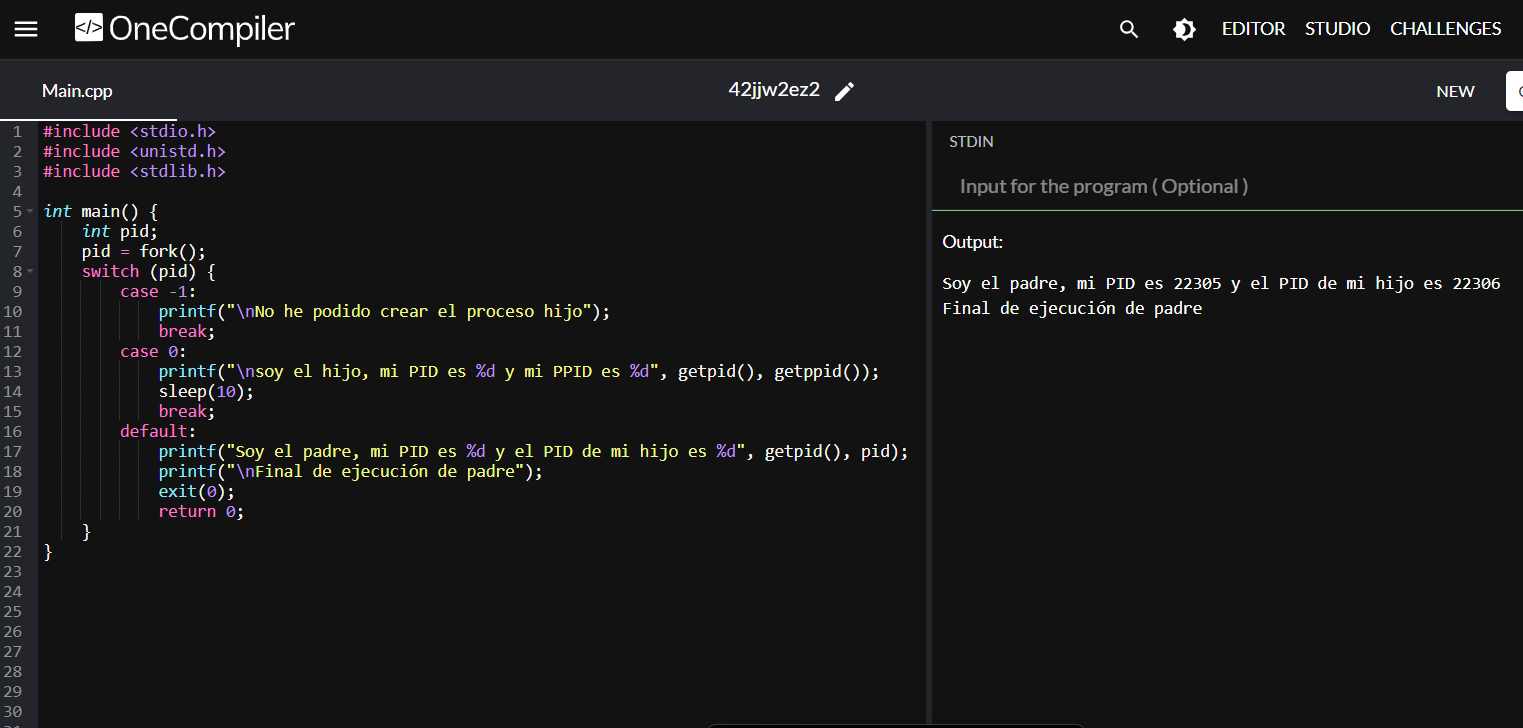
1. ****

Ilustración 8 Proceso hijo con espera.

1. **¿Cuál es el PPID del proceso hijo?**
2. El PPID (Parent Process ID) del proceso hijo es el PID del proceso padre. En la ejecución, el PID del proceso padre es 22305. Por lo tanto, el PPID del proceso hijo será 22305.
3. **¿En qué se diferencia el resultado con el código 3.1.4?**
4. En el código, el proceso padre imprime su mensaje de finalización y termina inmediatamente después de crear el proceso hijo. No espera a que el hijo termine su ejecución. Esto se observa en la llamada a exit(0) justo después del mensaje del padre.  
   El proceso hijo imprime su mensaje y luego duerme por 10 segundos (sleep(10)). Como el padre no espera, su mensaje de finalización se imprime antes de que el hijo termine de dormir.
5. **Modifique el código para que el proceso padre imprima el mensaje de finalización de su ejecución 10 segundos más tarde que el proceso hijo.**

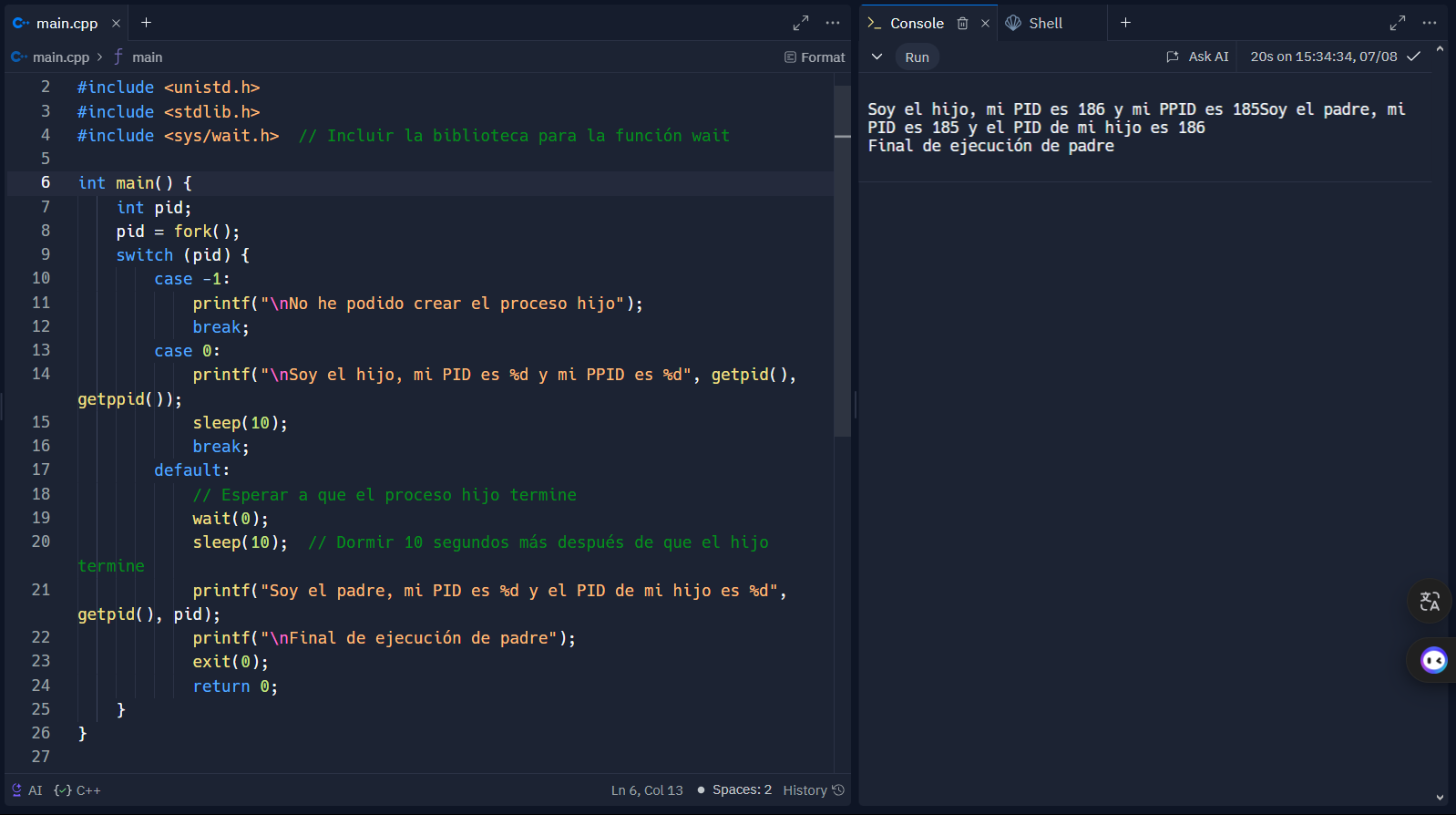


Ilustración 9 Mensaje de finalización 10 segundos más tarde que el proceso hijo.

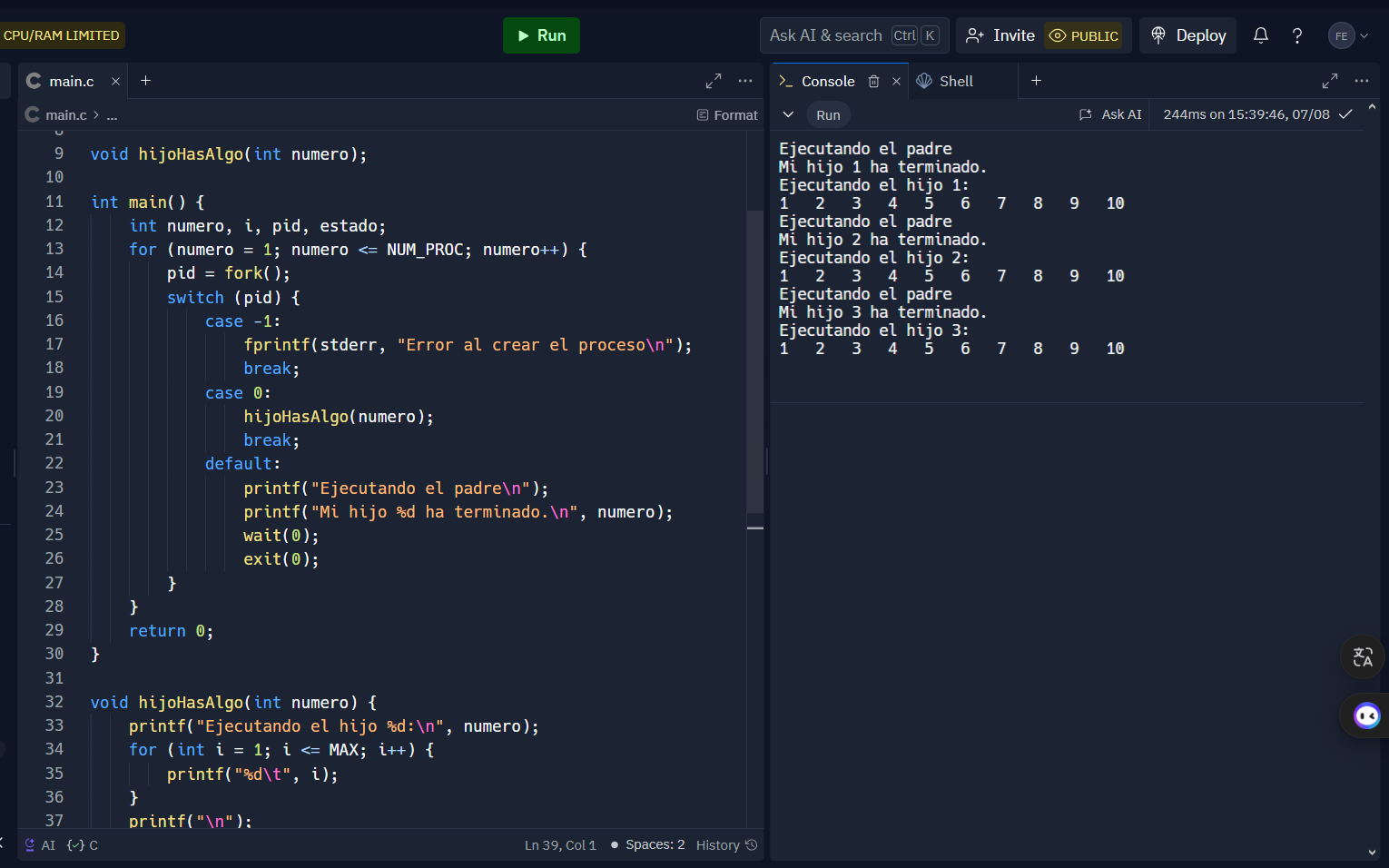
**3.1.6 Creación de un proceso padre y de tres procesos hijos. Cada uno imprime en pantalla del 1 al 10.**

Ilustración 10 Proceso padre y de tres procesos hijos

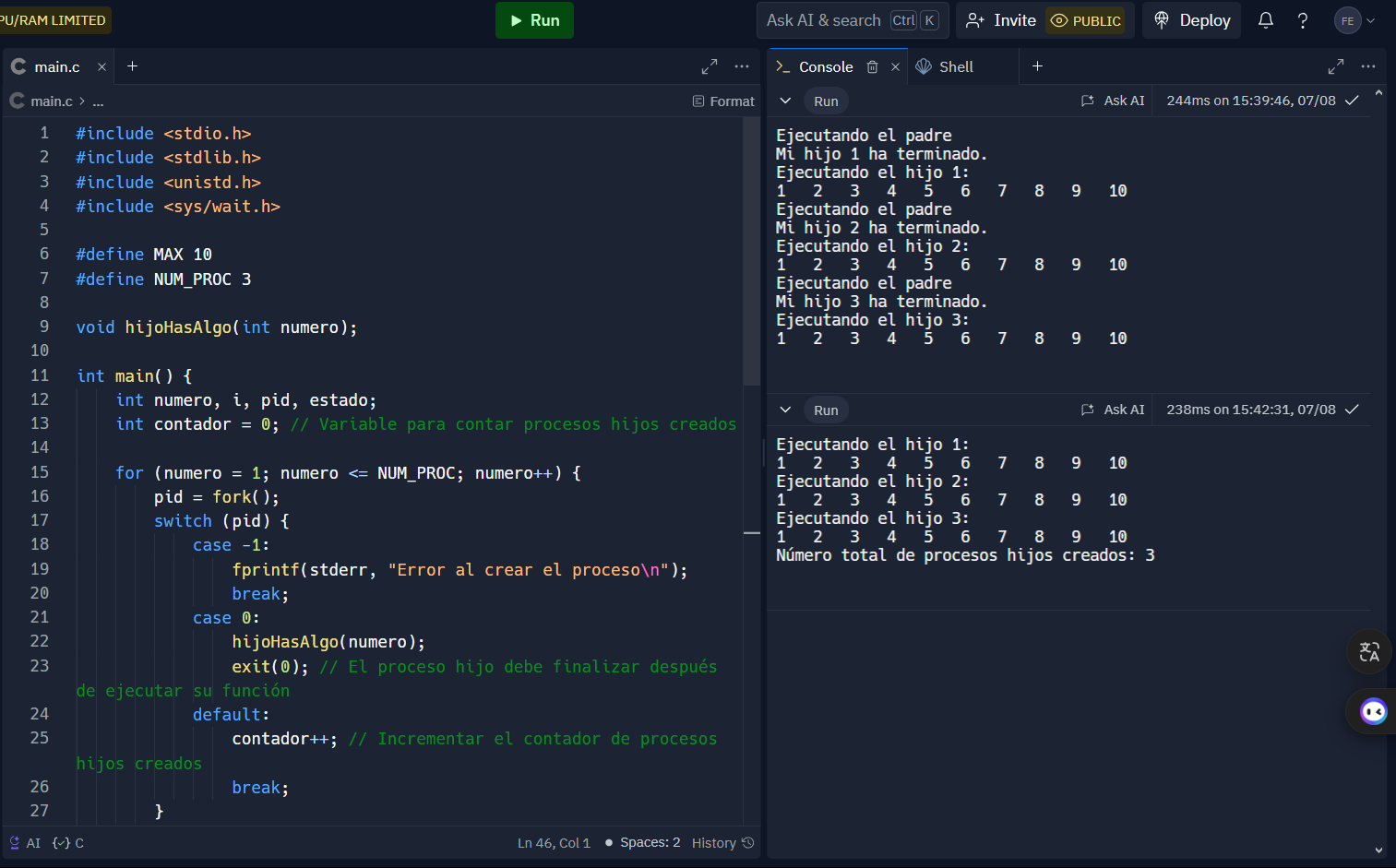
1. ¿Qué proceso **terminará** primero? ¿Por qué?
2. Para determinar cuál proceso terminará primero, debemos considerar que los procesos se ejecutan en paralelo y compiten por recursos de CPU. Sin embargo, no podemos predecir con certeza cuál será el orden exacto de finalización debido a la naturaleza concurrente de los procesos.
3. El orden de finalización puede variar según el sistema operativo y la planificación de procesos. Factores como la asignación de recursos, la carga del sistema y la sincronización afectarán el resultado.
4. **Modificar el código para que aparezca al final en pantalla el número de procesos hijos que se ejecutaron.**
5. ****

Ilustración 11 Código que imprime número el de procesos hijos que se ejecutaron.

## 3.1.7 Ejecución concurrente de procesos

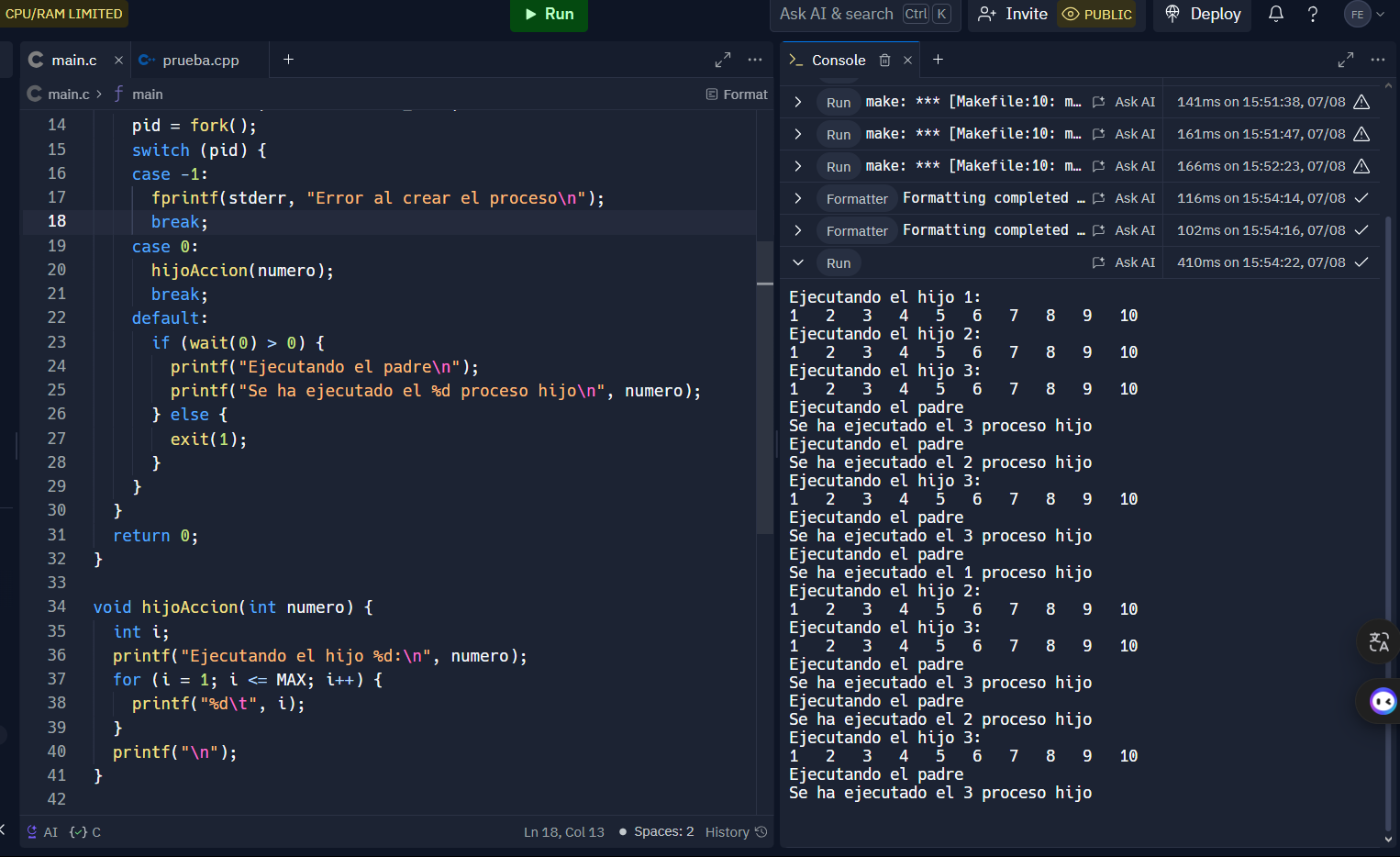
1. **¿Qué salida produce?**
2. ****

Ilustración 12 Ejecución concurrente de procesos

**¿Cuál es la diferencia con el código 3.1.6?**En este código, se crean tres procesos hijos utilizando fork(), y cada uno de ellos imprime los números del 1 al 10. Sin embargo, no se muestra el número total de procesos hijos ejecutados al final.

Para modificar el código y mostrar el número de procesos hijos ejecutados, podemos agregar una variable contador y aumentarla cada vez que se crea un proceso hijo. Luego, al final, imprimimos el valor de ese contador.

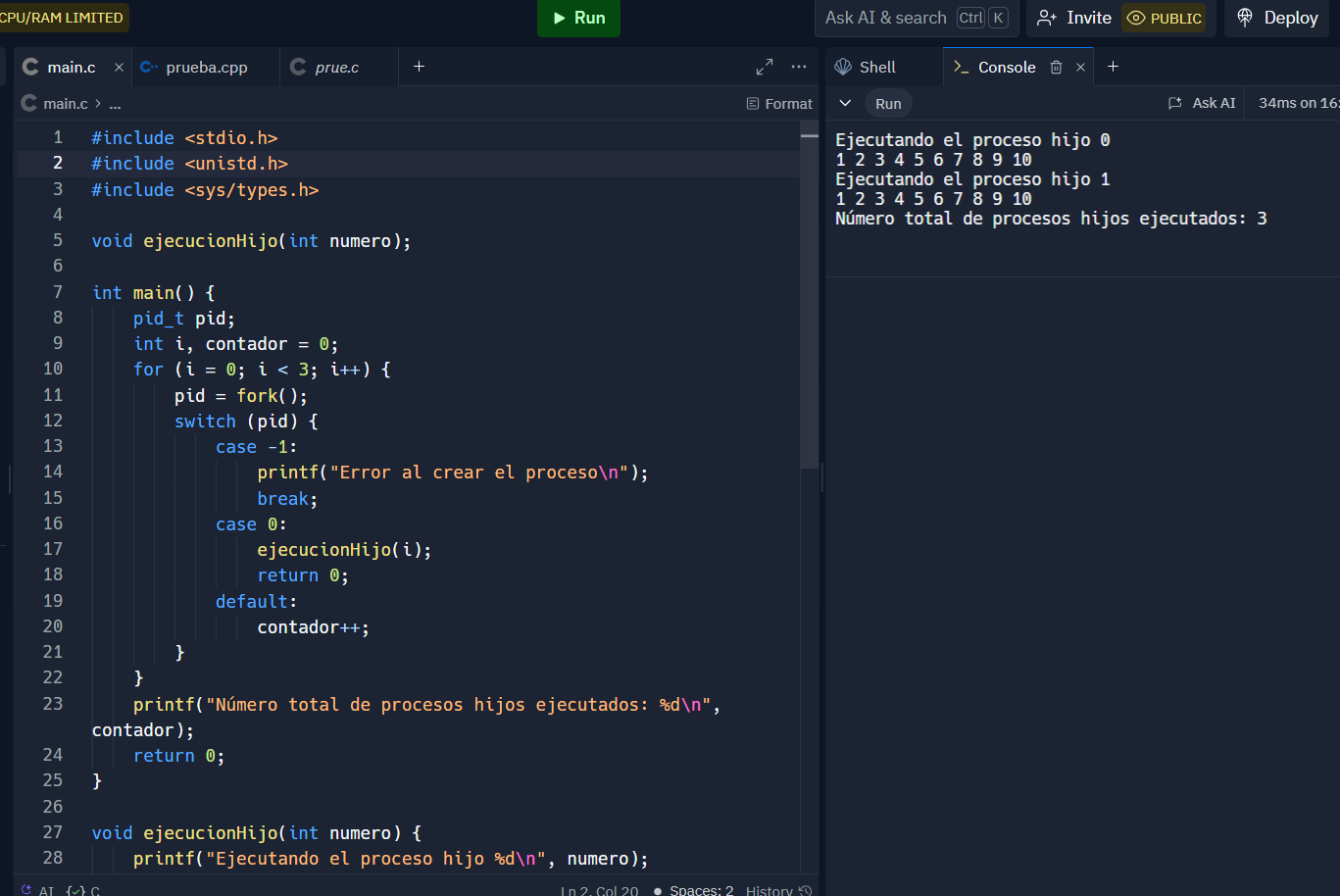
1. **Modificar el código para que aparezca al final en pantalla el número de procesos hijos que se ejecutaron.**
2. 

Ilustración 13 Código para que aparezca al final en pantalla el número de procesos hijos

## 3.2 USO DE EXEC

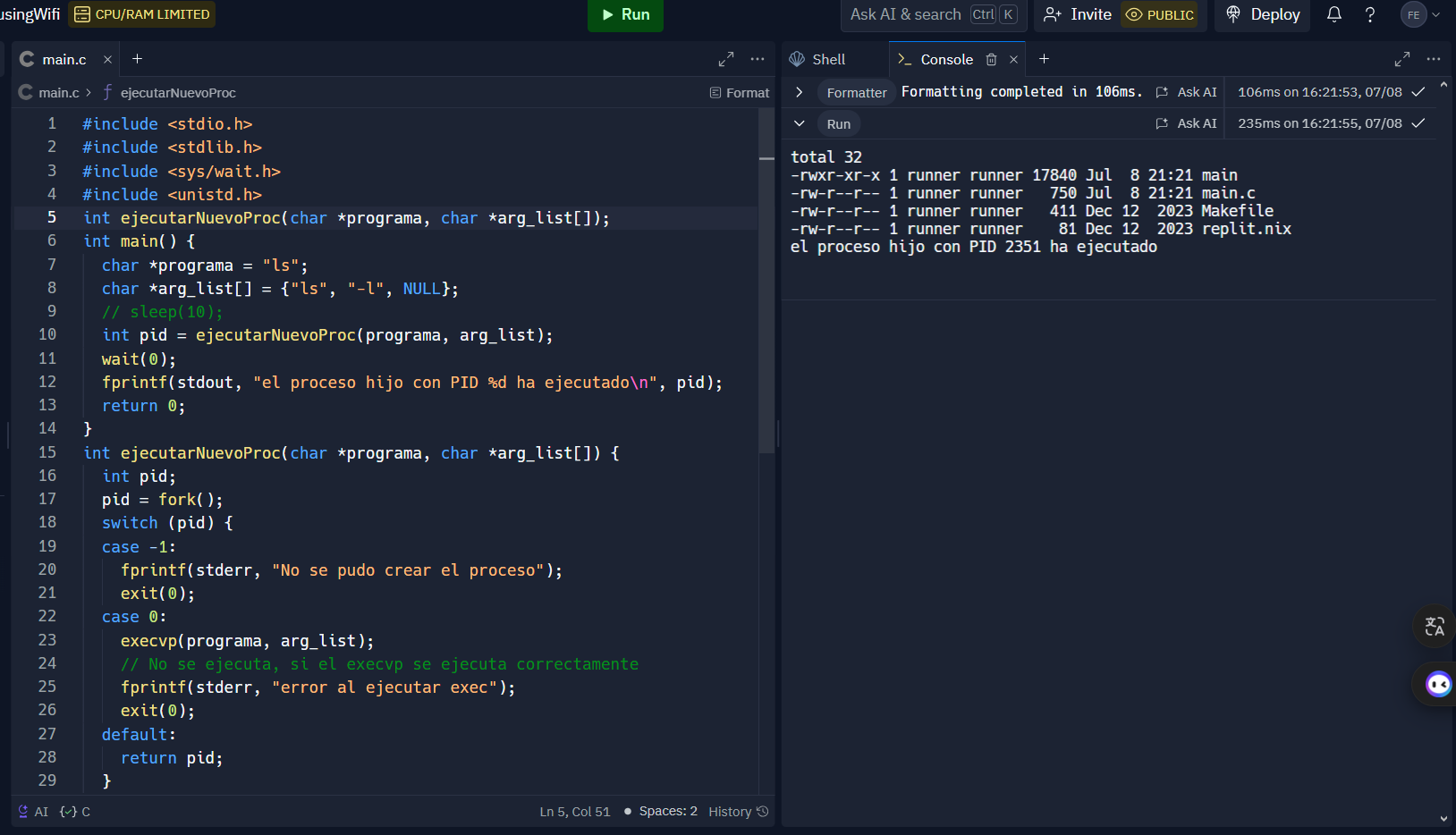
1. ¿Qué información se despliega al ejecutar el código?
2. 

Ilustración 14 Uso de exec

1. **¿Qué sucede si se cambia la función wait por sleep?**

 Si reemplazas wait(NULL) por sleep, el proceso padre simplemente se detendrá durante el número de segundos especificado en sleep y luego continuará su ejecución sin esperar realmente a que el proceso hijo termine.

 Si el proceso hijo termina antes de que el tiempo de sleep haya pasado, el proceso padre no lo sabrá y continuará durmiendo.

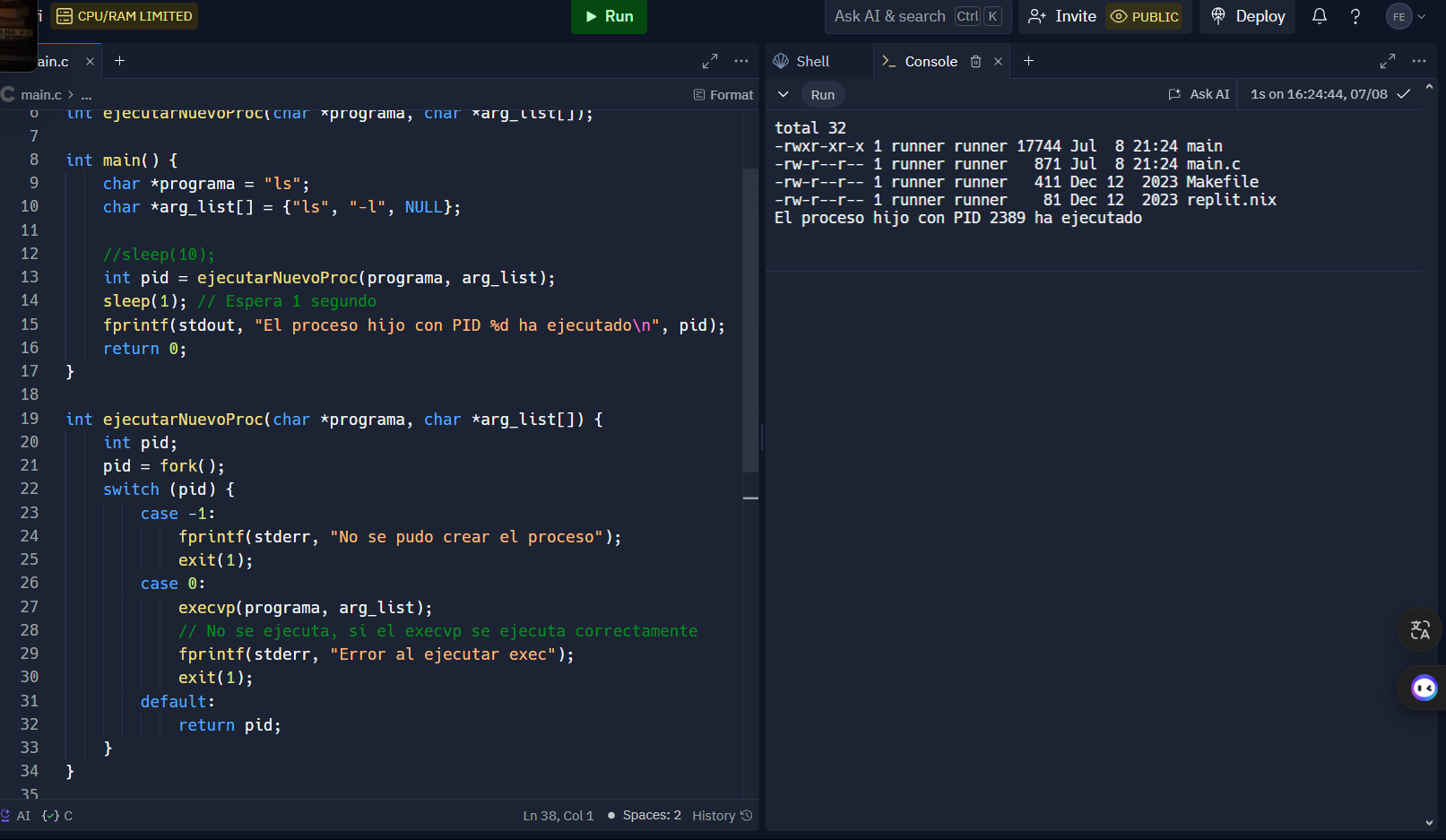
1.  Si el proceso hijo no ha terminado cuando el proceso padre se despierta, el proceso padre continuará su ejecución de todas formas, lo que puede llevar a comportamientos indeterminados si el programa depende del resultado del proceso hijo.
2. 

Ilustración 15 Se ha cambiado la función wait por sleep

## 3.2.1 Creación de un proceso zombie

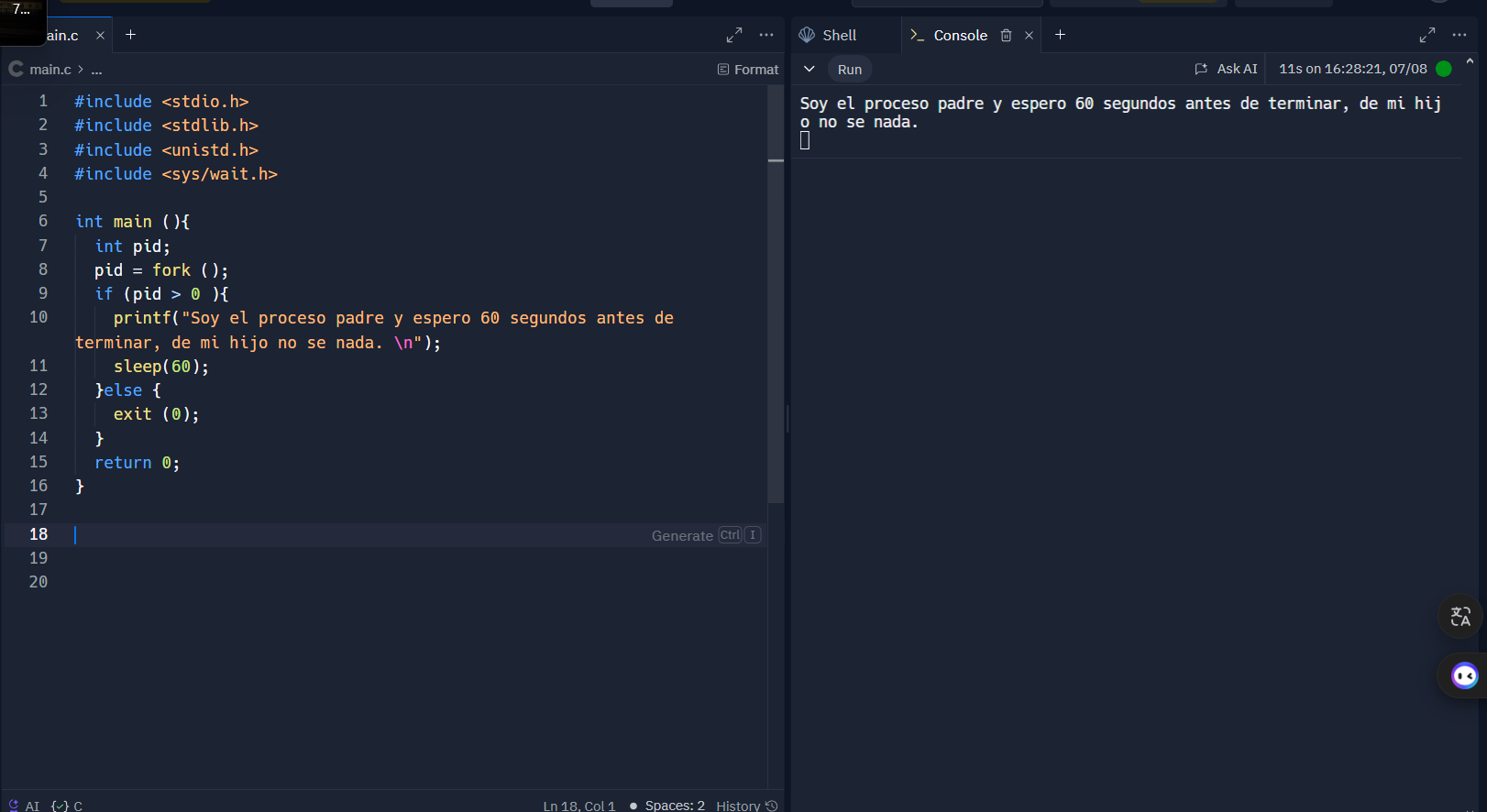
1. **Ejecutar el código en segundo plano**
2. ****

Ilustración 16 Creación de un proceso zombie

1. **Ejecutar el comando top y verificar que existe un proceso zombie**
2. **¿Cómo solucionaría el problema?**
3. Cuando el proceso hijo termina después de llamar a exit(0), se convierte en un "proceso zombie". Un proceso zombie es aquel cuyo proceso padre aún no ha llamado a wait() o waitpid() para obtener su estado de salida. Aunque el proceso hijo ha terminado de ejecutarse, su entrada en la tabla de procesos aún existe hasta que el proceso padre obtenga su estado de salida.
4. Para evitar que los procesos hijos se conviertan en zombies, el proceso padre debe manejar adecuadamente la finalización de los procesos hijos. Esto se logra utilizando la función wait() o waitpid() después de que el proceso hijo haya terminado. Estas funciones permiten al proceso padre esperar y obtener el estado de salida del proceso hijo, lo que elimina la entrada del proceso

zombie de la tabla de procesos del sistema.

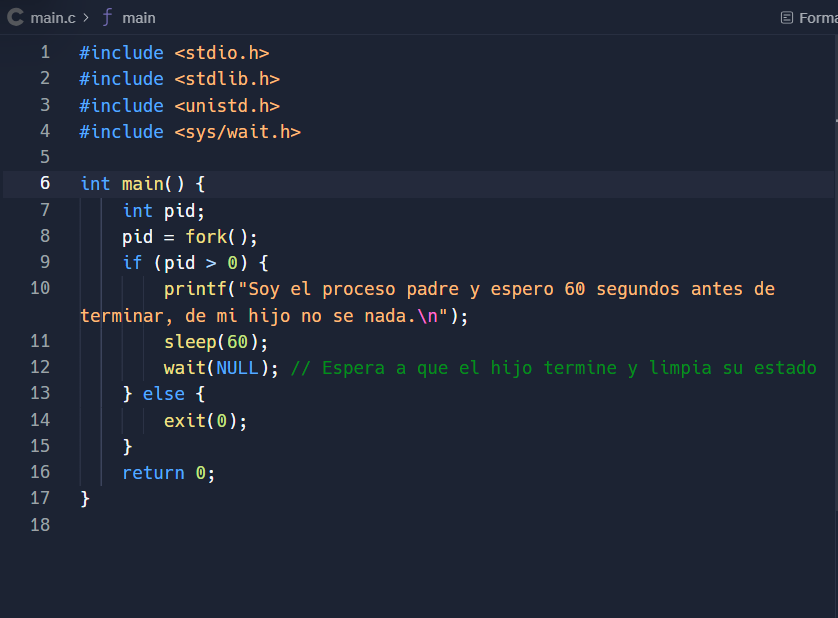
1. ****

Ilustración 17 Código que soluciona el problema planteado

## 3.2.2 Uso de exec

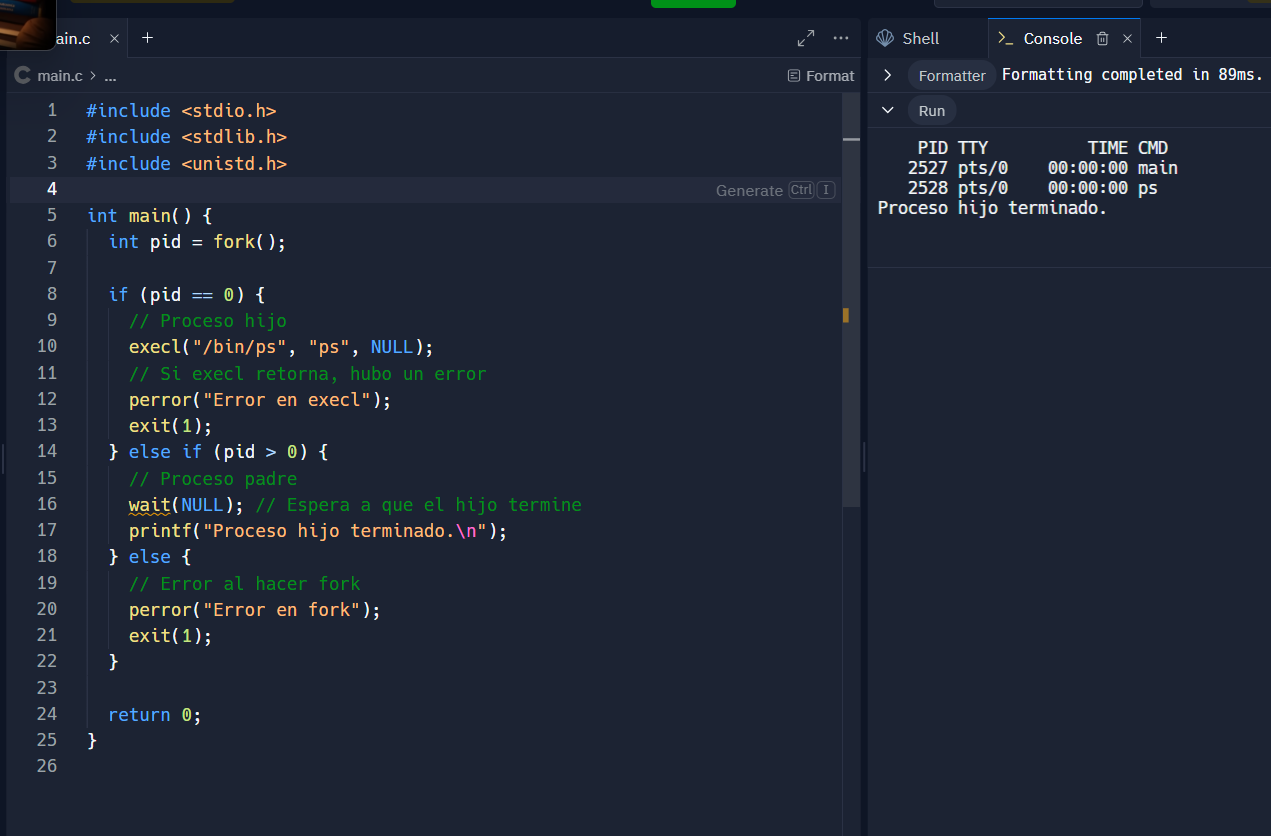
1. Crear un **proceso** hijo que liste los procesos del sistema usando execl
2. 

Ilustración 18 Proceso hijo que liste los procesos del sistema usando execl

## 3.2.3 Crear un proceso hijo y colocarlo dentro de un bucle infinito.

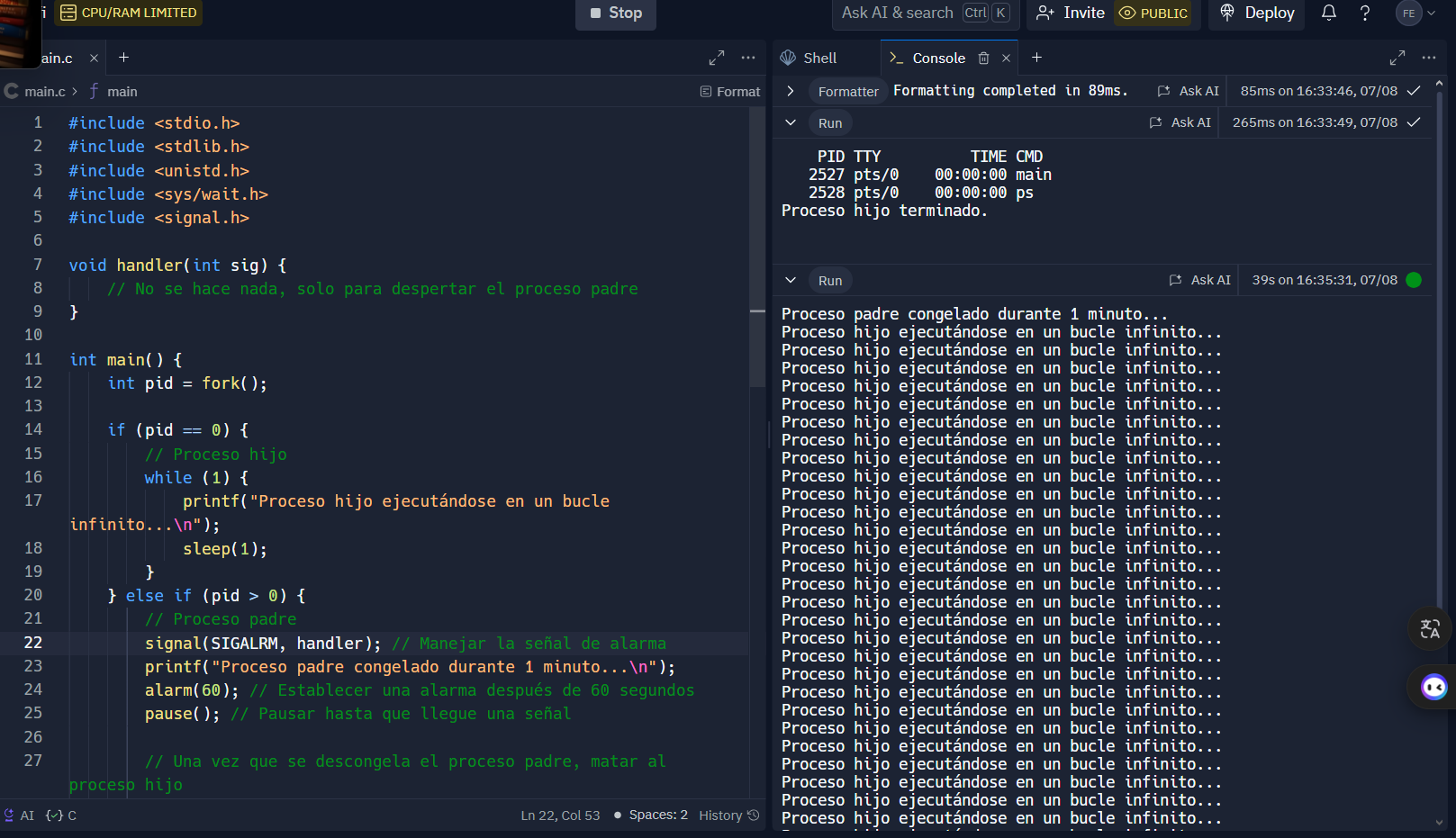
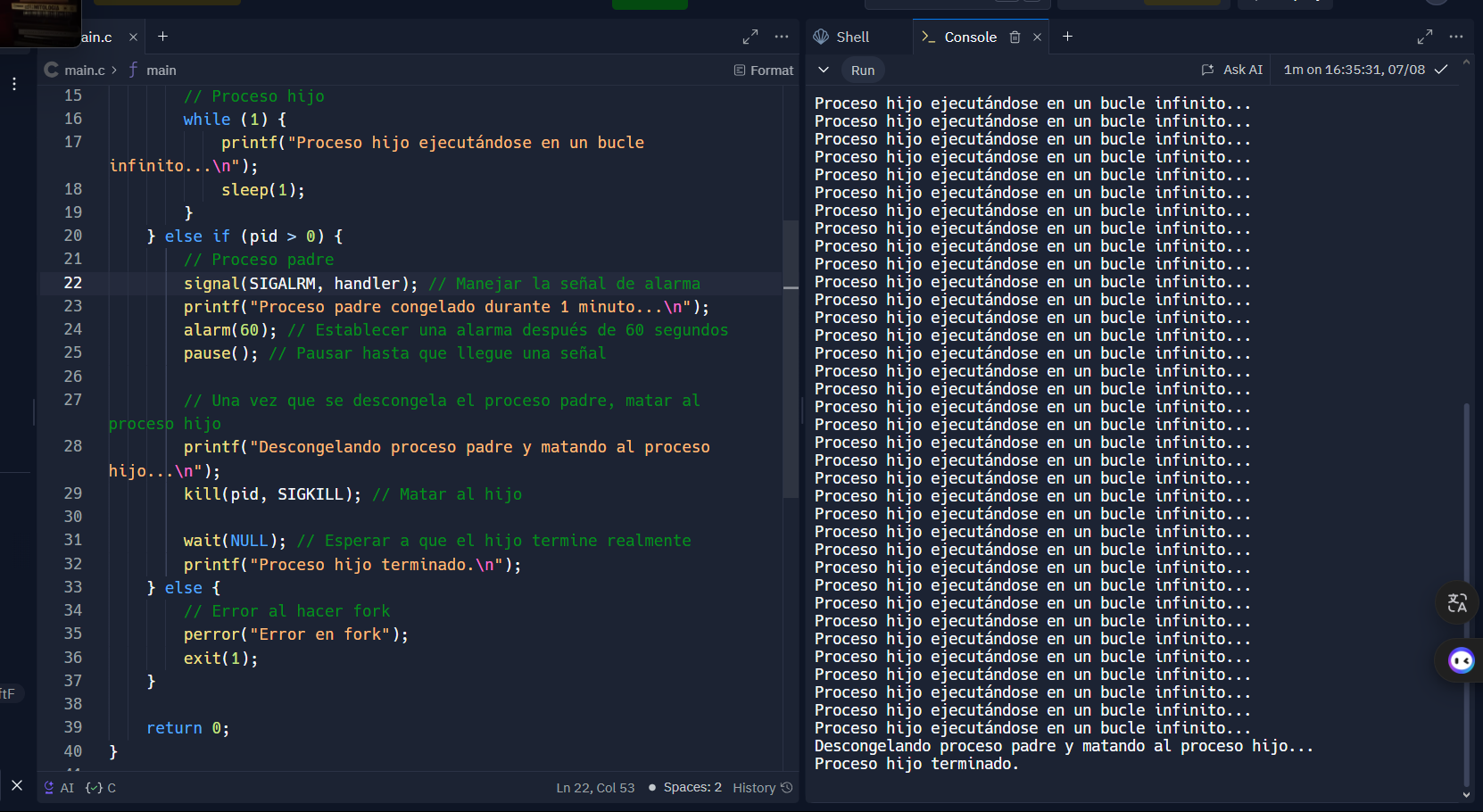
1. **El proceso padre se congelará durante 1 minuto.**
2. **Una vez que se descongele el proceso padre se deberá matar al proceso hijo.**
3. 
4. 

Ilustración 19 Creación de un proceso hijo colocado dentro de un bucle infinito.

# CONCLUSIONES Y RECOMENDACIONES

* Se pudo comprender la sincronización entre procesos, como esperar a que un proceso hijo termine antes de que el padre continúe, es crucial para evitar problemas como los procesos zombies (procesos hijos que han terminado pero cuyo estado no ha sido recogido por el padre).
* Aprendimos a manejar adecuadamente la creación y finalización de procesos utilizando funciones como wait(), waitpid() y señales como SIGKILL para asegurar que los recursos del sistema se utilicen de manera eficiente y segura.
* Cuando se trabaja con múltiples procesos, es crucial diseñar y desarrollar aplicaciones que sean eficientes y seguras. Esto implica pensar en la sincronización, la comunicación entre procesos y la gestión adecuada de recursos compartidos.
* Es esencial manejar adecuadamente la creación y finalización de procesos utilizando funciones como wait(), waitpid() y señales como SIGKILL para asegurar que los recursos del sistema se utilicen de manera eficiente y segura.práctica.

# BIBLIOGRAFÍA

[1] A. S. Tanenbaum and H. Bos, Modern Operating Systems, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014.

[2] W. Stallings, Operating Systems: Internals and Design Principles, 9th ed. Boston, MA, USA: Pearson, 2017.

[3] J. López y A. García, "Gestión de Procesos en Sistemas Operativos Unix," Revista Iberoamericana de Informática, vol. 21, no. 3, pp. 45-58, Sep. 2010.