

FORK()

Sistemas Operativos

**VIVIANA GÓMEZ, GABRIEL JARAMILLO, ROBERTH
MÉNDEZ, LUZ SALAZAR, GUDEN SILVA**



Historia

- El concepto fue referenciado por un artículo de Melvin Conway en 1962.

Creación

Fue originado a finales de los 60's por Ken Thompson y Dennis Ritchie cuando lo desarrollaron para los primeros sistemas UNIX.

Ken Thompson y Dennis Ritchie



¿Que es fork()?



¿Para que sirve?

- Permite ejecutar múltiples procesos en paralelo.
- Permite concurrencia (más eficiente)

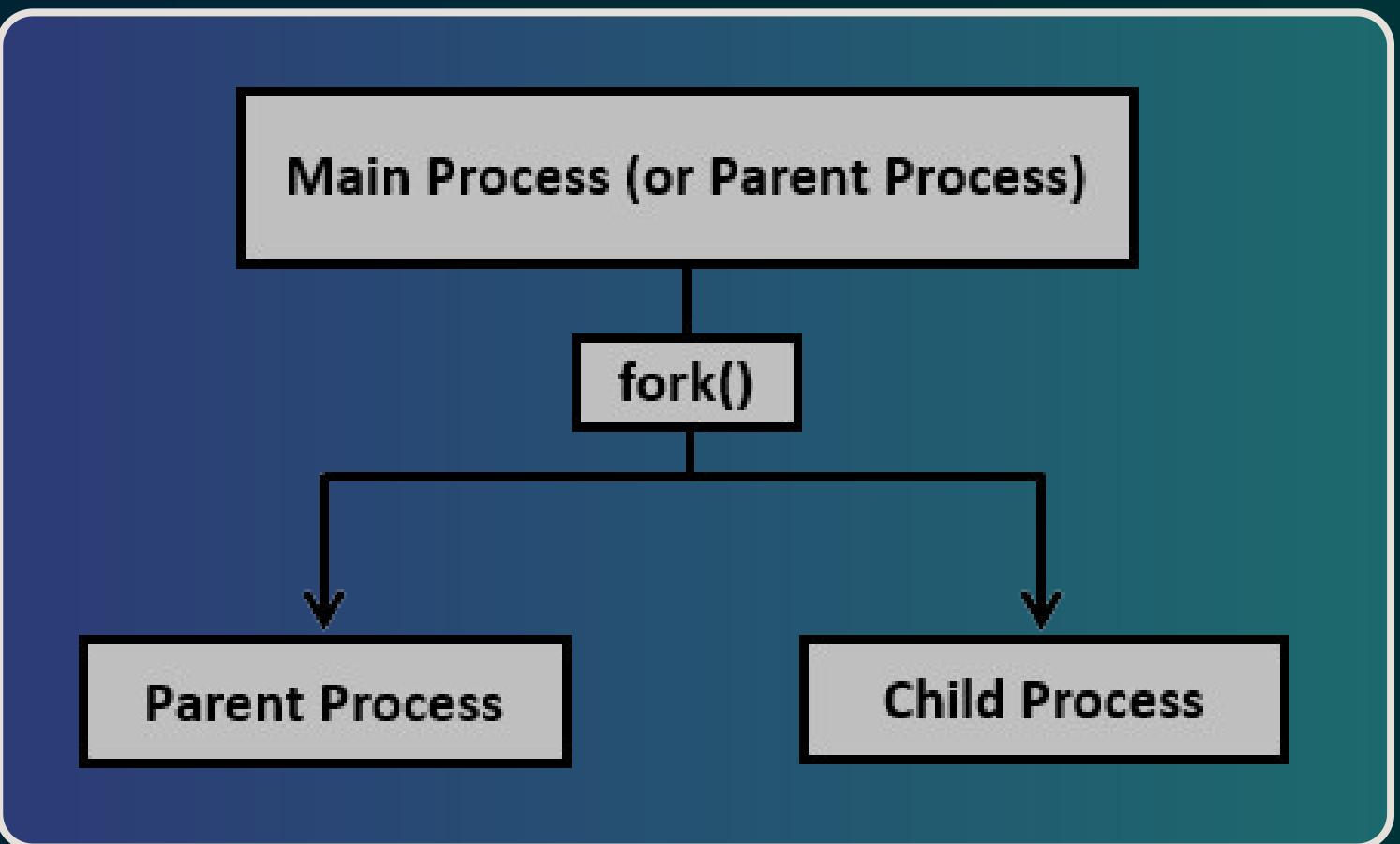
Llamada al sistema que permite crear un nuevo proceso a partir de un proceso existente.

Como funciona

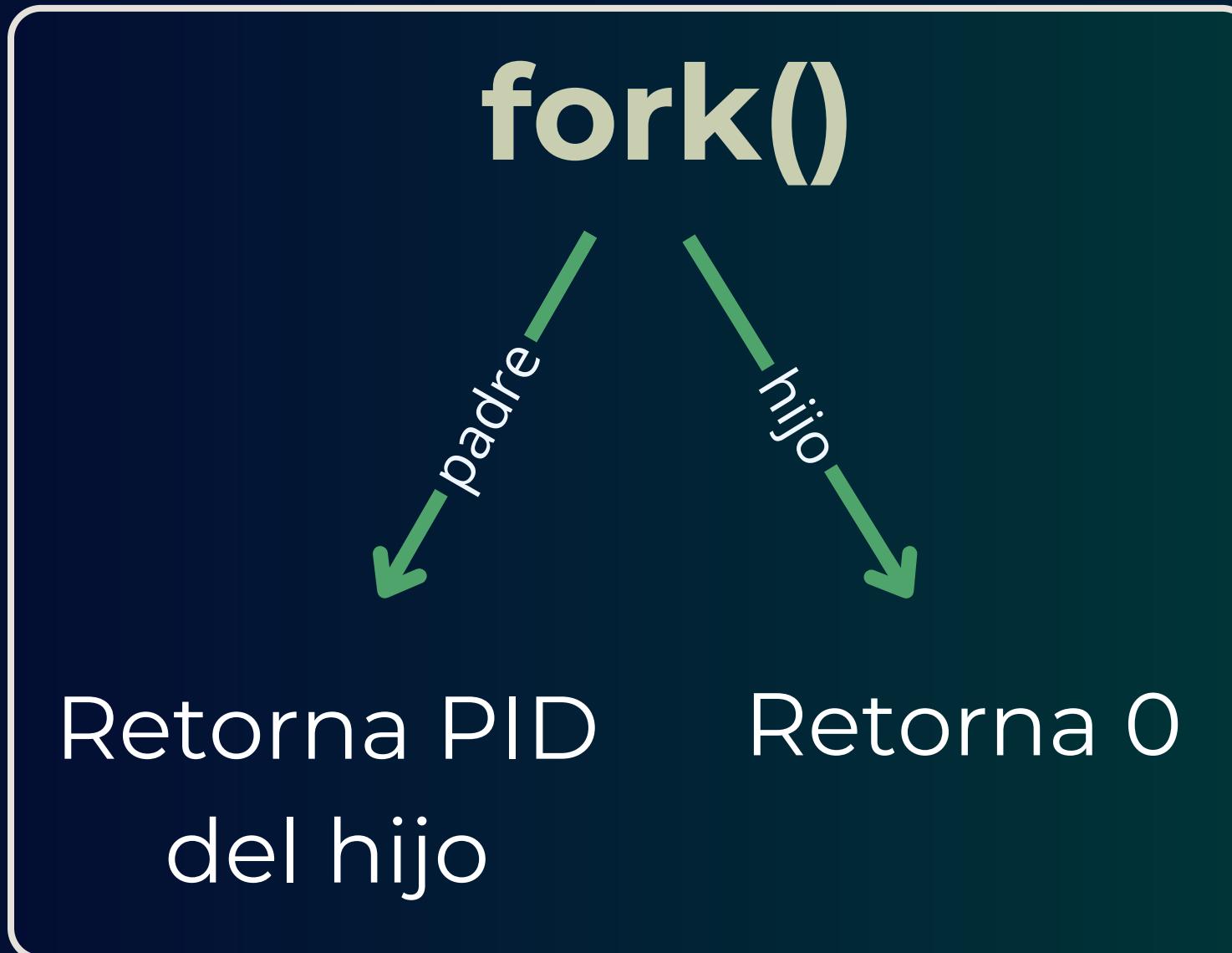
fork() crea un nuevo proceso (proceso hijo) a partir del proceso actual (proceso padre)

Hijo

- Copia casi exacta del padre.
 - Hereda recursos del padre.
 - Tiene su propio PID (Process ID).
 - Su PPID (Parent Process ID) es el PID del proceso padre.
-
- S.O. decide cuál proceso se ejecuta primero.
 - Ambos procesos continúan ejecutándose concurrentemente después de fork().
 - Los procesos padre e hijo tienen diferentes espacios de direcciones.



Valores de retorno



Exitoso

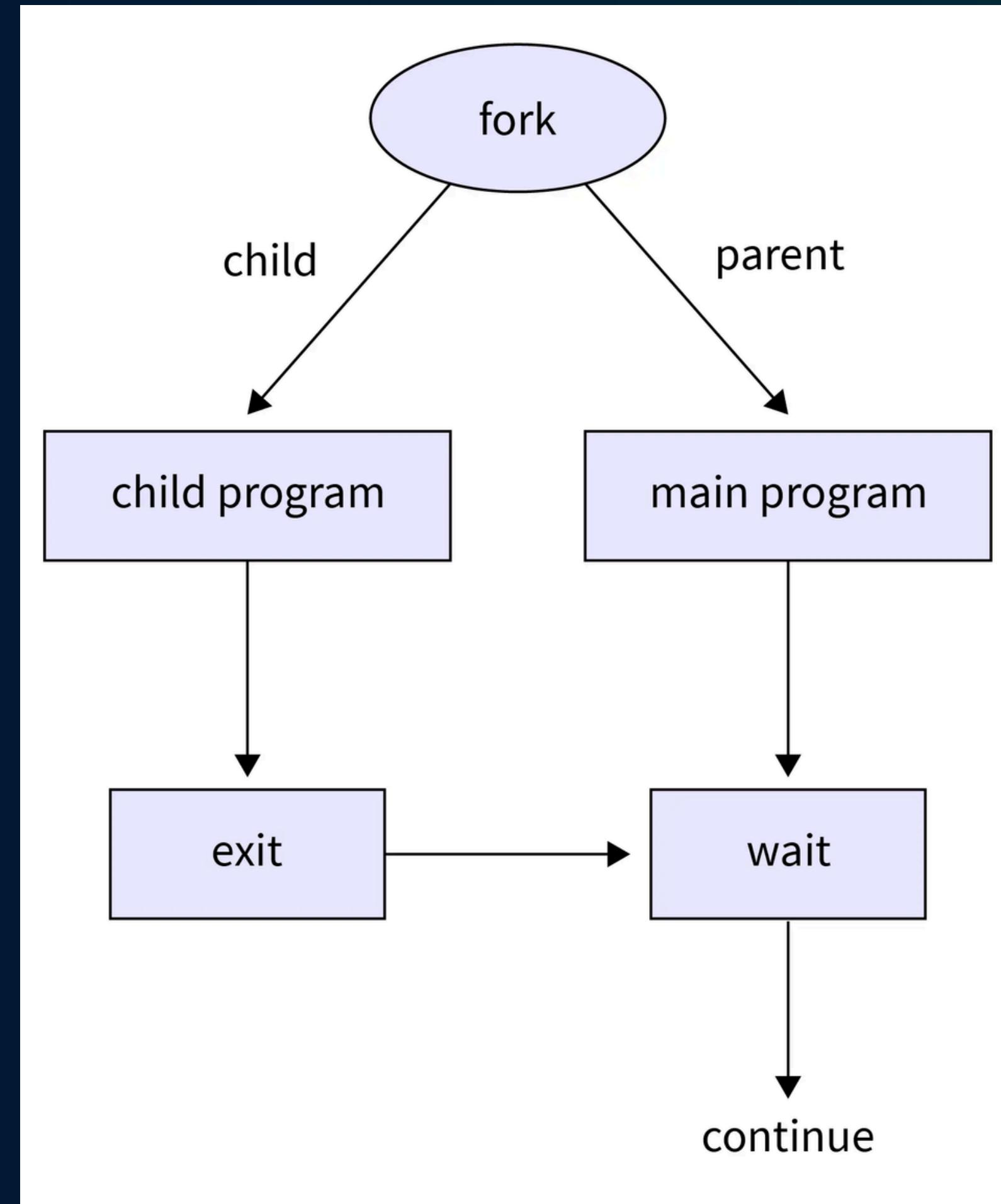
Padre: Retorna PID del hijo.

Hijo: Retorna 0.

No exitoso

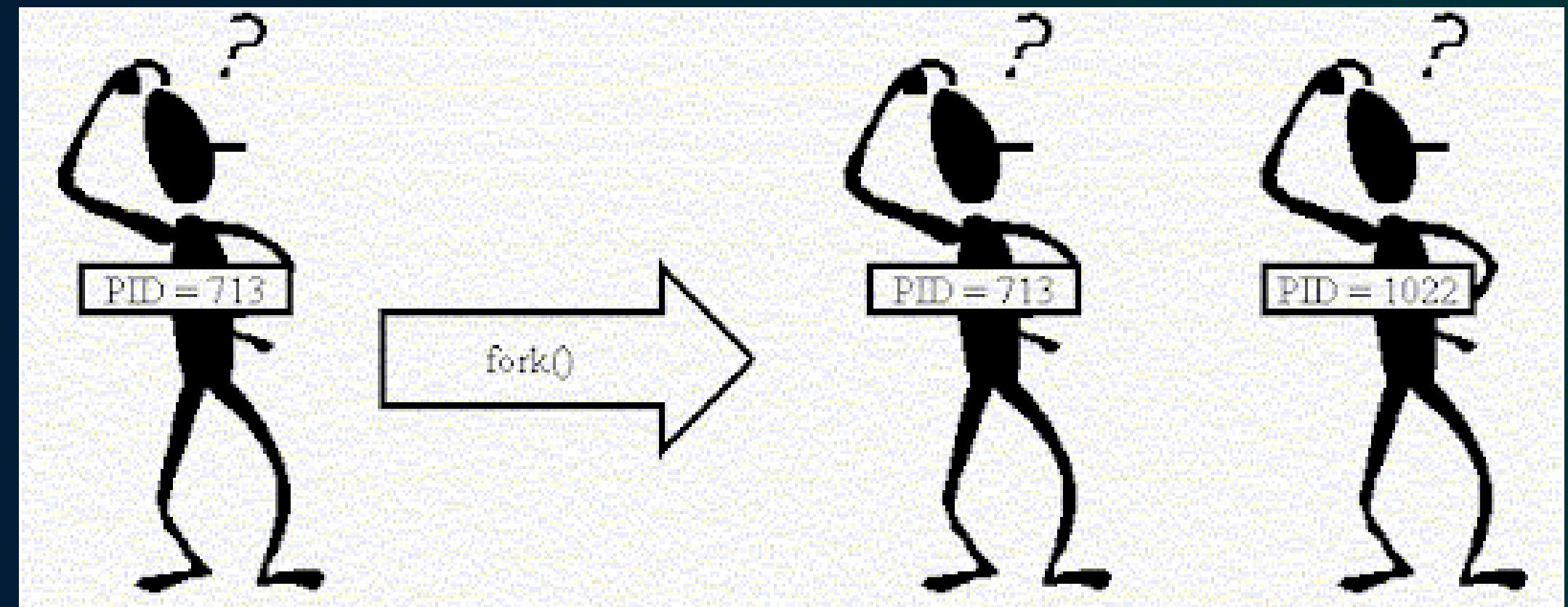
Retorna -1 y el proceso hijo no se crea.

- Falta de memoria (ENOMEM).
- Límite de procesos alcanzado (EAGAIN).



¿Qué hereda el hijo?

-
- Código del programa.
- Variables y datos
- Descriptores de archivo
- Variables de entorno.
- Señales y controladores de señales (excepto las en estado pendiente)
- Identificador de usuario y grupo



Identificadores de proceso (PID) y de padre (PPID) pasan a ser valores nuevos, no se heredan

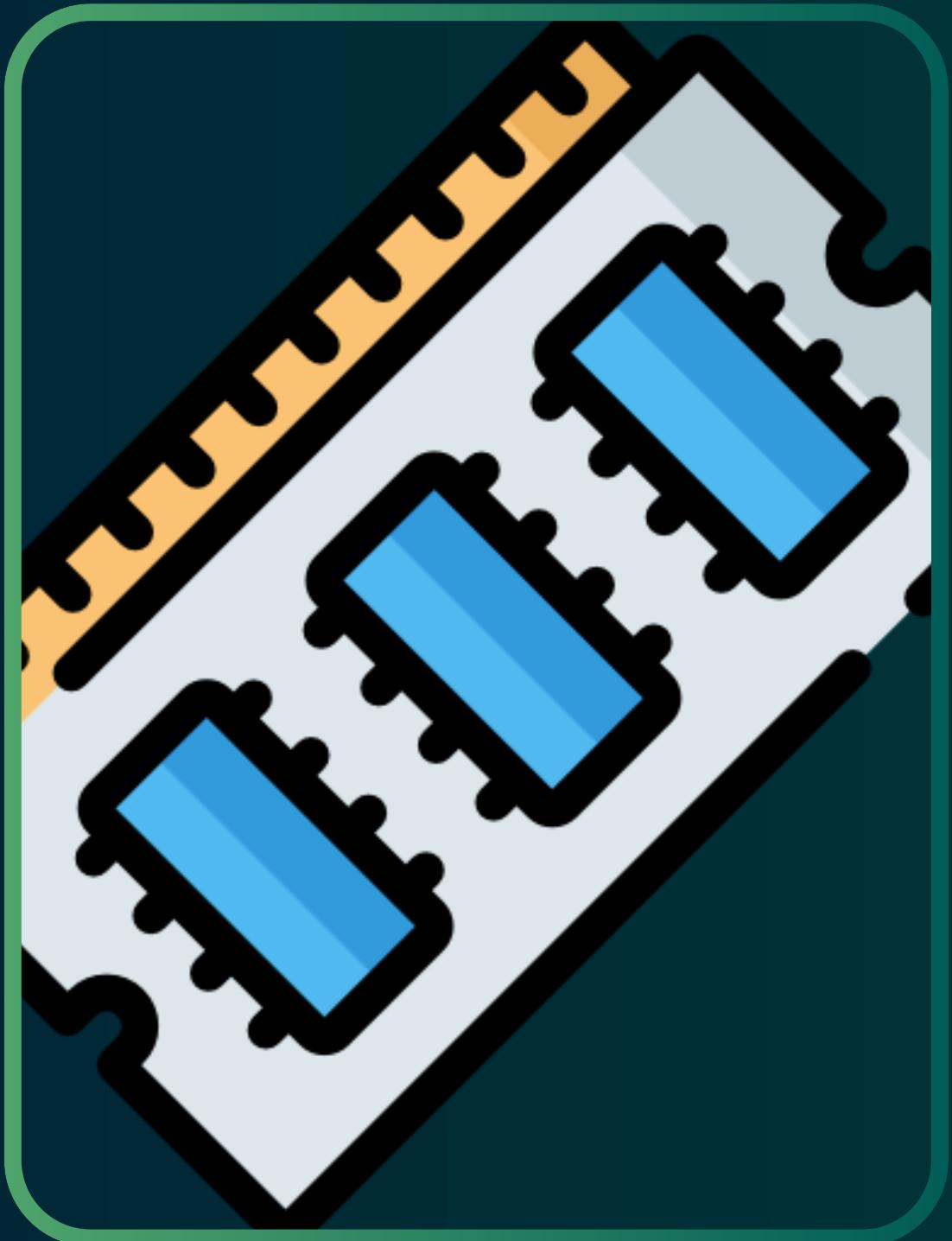


ESPACIO DE MEMORIA DEL HIJO

Se crea una copia del espacio de memoria del padre que incluye:

- ◆ Segmento de código (el programa en sí).
- ◆ Segmento de datos (variables globales y estáticas).
- ◆ Segmento de pila (donde se guardan variables locales y llamadas a funciones).
- ◆ Segmento de heap (memoria dinámica con malloc(), new, etc.).

Hacer la copia es lento y consume mucha memoria, para solucionar esto se hace un Copy-On-Write (COW).





Copy-On-Write

ACCIÓN	PADRE	HIJO
Inicialmente	Misma memoria compartida, solo lectura 	Misma memoria compartida, solo lectura 
El hijo solo lee	Sin copias	Sin copias
El hijo escribe	Sin cambios	Copia solo la página modificada



fork() vs vfork() vs pthread_create()

	fork()	vfork()	pthread_create()
¿Crea un nuevo proceso?	Si	Si	No (crea un hilo el mismo proceso)
¿Copia memoria?	Si, Copy-On-Write	No (el hijo usa la misma memoria del padre hasta que llama exec())	No (Comparten memoria)
¿Es más rápido que fork() ?	-	Más rápido	No necesita COW
¿Usa Copy-On-Write?	Si	Mucho más rápido	-

Problemas comunes y soluciones



Condición de Carrera

Se produce cuando dos o más procesos acceden a una variable compartida al mismo tiempo y el resultado final depende del orden de ejecución de estos accesos.

Sincronizar Subprocesos

Técnicas para controlar el acceso a recursos compartidos:

- Mutex
- Semáforos
- Monitores

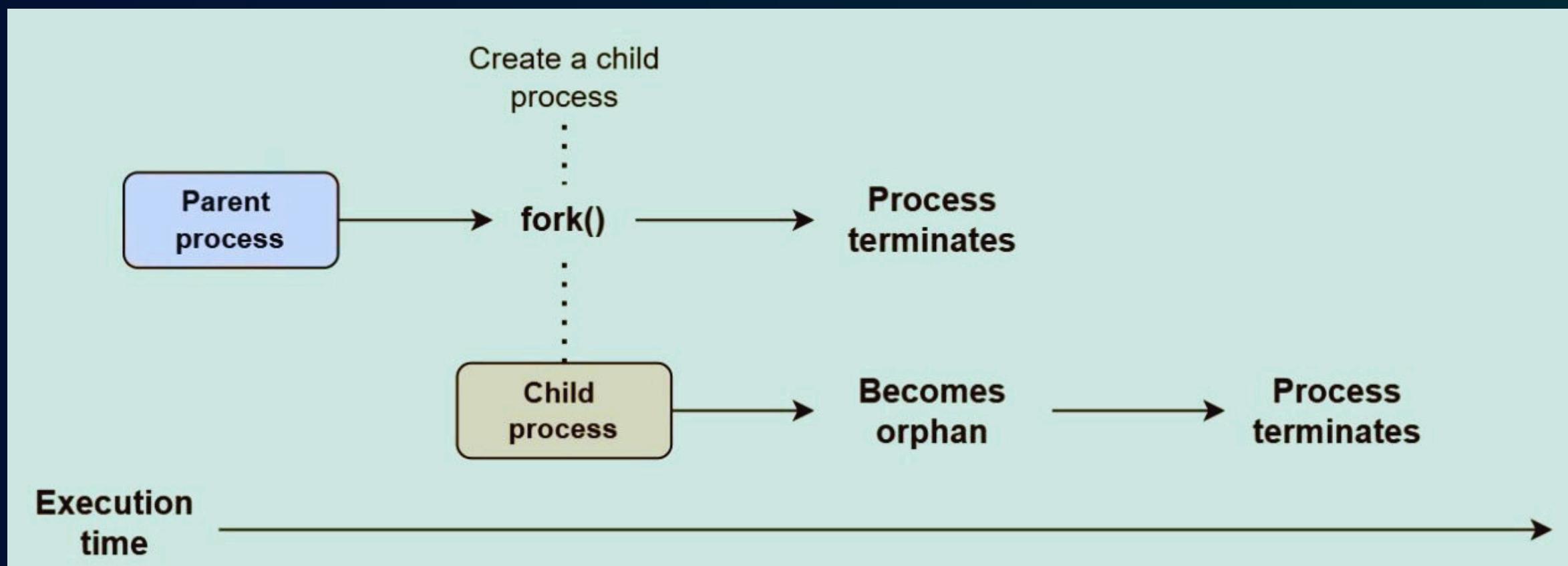
Ejemplo

Tiempo	Proceso Padre	Proceso Hijo
t1	Lee counter = 0	
t2		Lee counter = 0
t3	Incrementa a 1	
t4		Incrementa a 1
t5	Escribe counter = 1	
t6		Escribe counter = 1
Final	counter = 1	counter = 1



Procesos Huérfanos

Es un proceso hijo que sigue realizando su ejecución, después de que el proceso padre ya ha finalizado y abandona la tabla de procesos sin esperar a que el proceso hijo finalice.



El **kernel** le asigna un nuevo proceso padre, generalmente el **proceso init.**



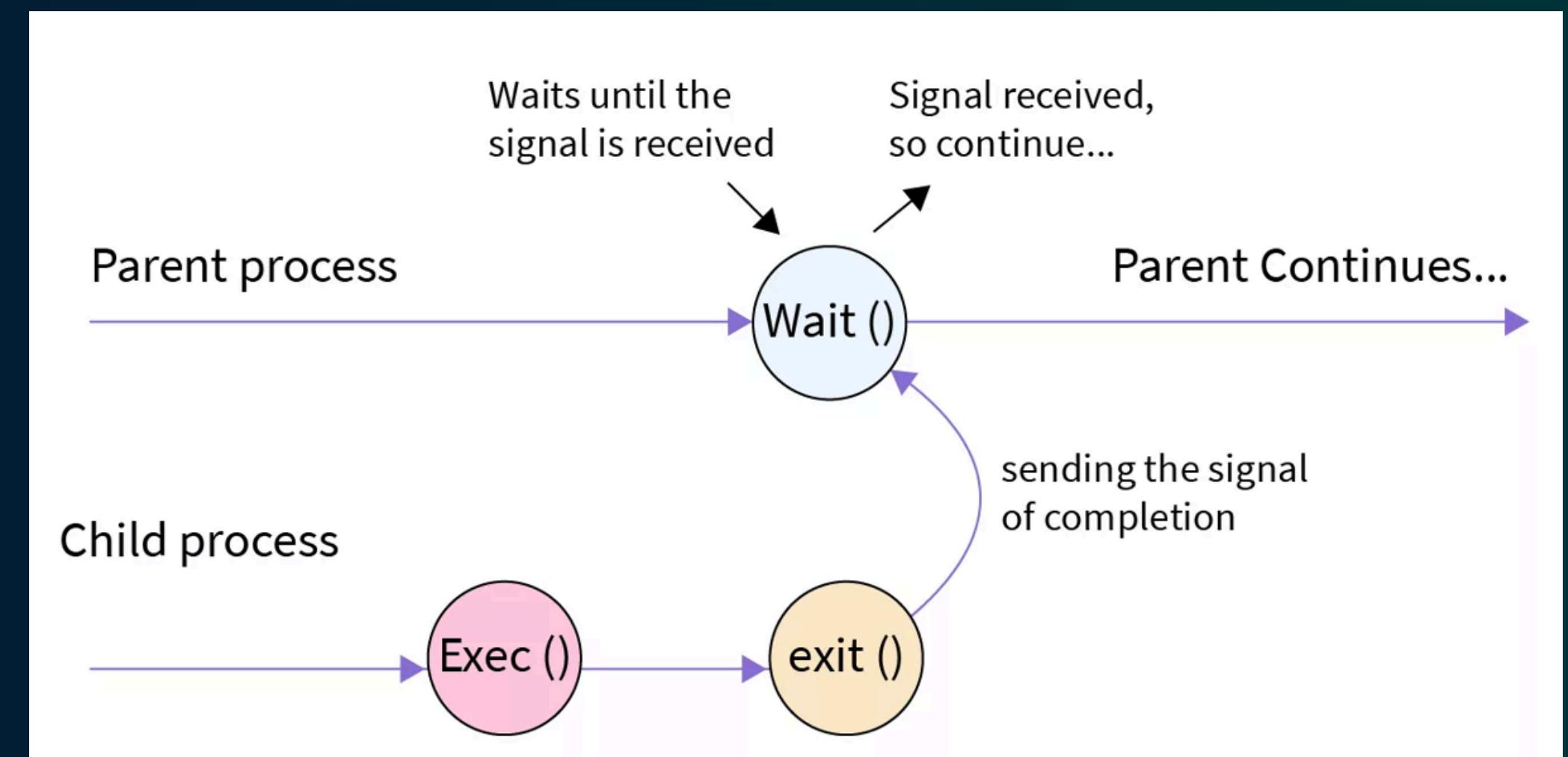
Problemas Comunes

Procesos Zombies

Es un **proceso finalizado** que ya no se ejecuta pero que sigue apareciendo en la **tabla de procesos**.

El proceso padre todavía tiene el **estado** del proceso marcado como **en ejecución**.

El proceso padre no llama a la función "wait()" para leer el estado de salida del proceso hijo.



Aplicaciones y Usos

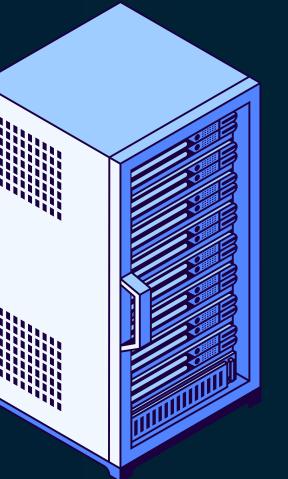
Aplicaciones

Tareas Múltiples



Realizar diferentes tareas al mismo tiempo mediante la ejecución múltiple de procesos.

Servidores Web



Creación de varios procesos para atender solicitudes de cada usuario de forma individual.

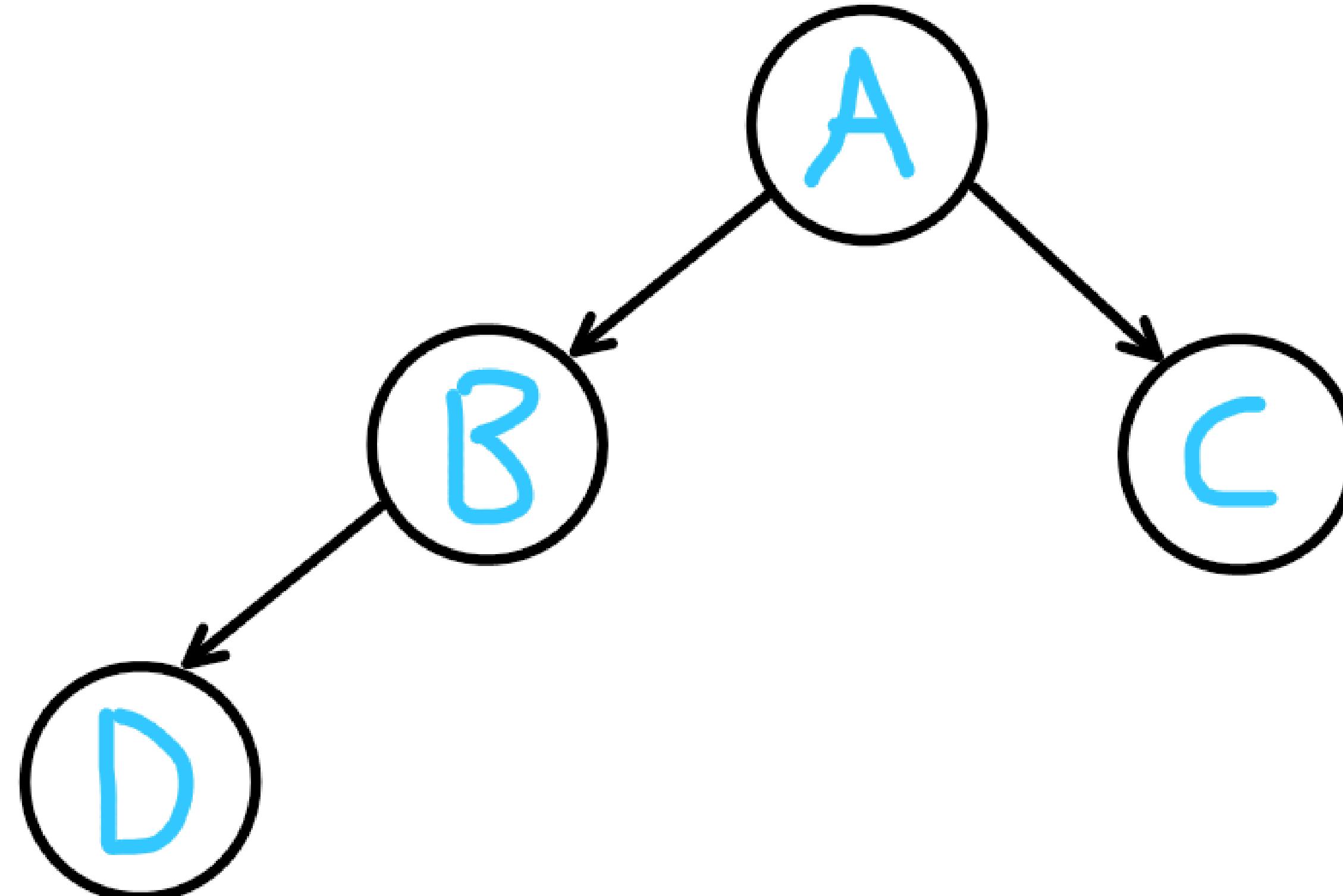
Entornos simulados



Crear procesos hijos y estudiar como interactúan entre ellos.

CÓDIGO DE EJEMPLO

CÓDIGO DE EJEMPLO



CÓDIGO DE EJEMPLO

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5
6 int main() {
7
8     printf("ÁRBOL DE PROCESOS\n");
9
10    pid_t pidB, pidC, pidD;
11
12    // Proceso padre - Proceso A
13    printf("Soy el proceso A, mi PID es %d y el PID de mi padre es %d\n", getpid(), getppid());
14
15    pidB = fork(); // Creo el proceso B
16
17    // Error al crear B
18    if (pidB < 0){
19        printf("Error al crear el B");
20        exit(1);
21    }
22}
```

CÓDIGO DE EJEMPLO

```
// Proceso B
if (pidB == 0){

    printf("Soy el proceso B, mi PID es %d y el PID de mi padre es %d\n", getpid(), getppid()); // Primer hijo de A

    // B crea el proceso D
    pidD = fork();

    if (pidD < 0){
        printf("Error al crear D");
        exit(1);
    }

    // Proceso D
    if (pidD == 0) {
        printf("Soy el proceso D, mi PID es %d y el PID de mi padre es %d\n", getpid(), getppid()); // Primer hijo de B
        while (1); // Mantener activo el proceso D
    }

    while (1); // Mantener activo el proceso B
}
```

CÓDIGO DE EJEMPLO

```
// Proceso A
if (pidB > 0){

    pidC = fork(); // A crea el proceso C
    if (pidC < 0){
        printf("Error al crear C");
        exit(1);
    }

    // Proceso C
    if(pidC == 0){

        printf("Soy el proceso C, mi PID es %d y el PID de mi padre es %d\n", getpid(), getppid()); // Segundo hijo de A
        while (1); // Mantener activo el proceso C
    }
}

while (1); // Mantener activo el proceso A
```

RESULTADOS

```
estudiante@NGEN211:~/Operativos/Fork$ ./test
```

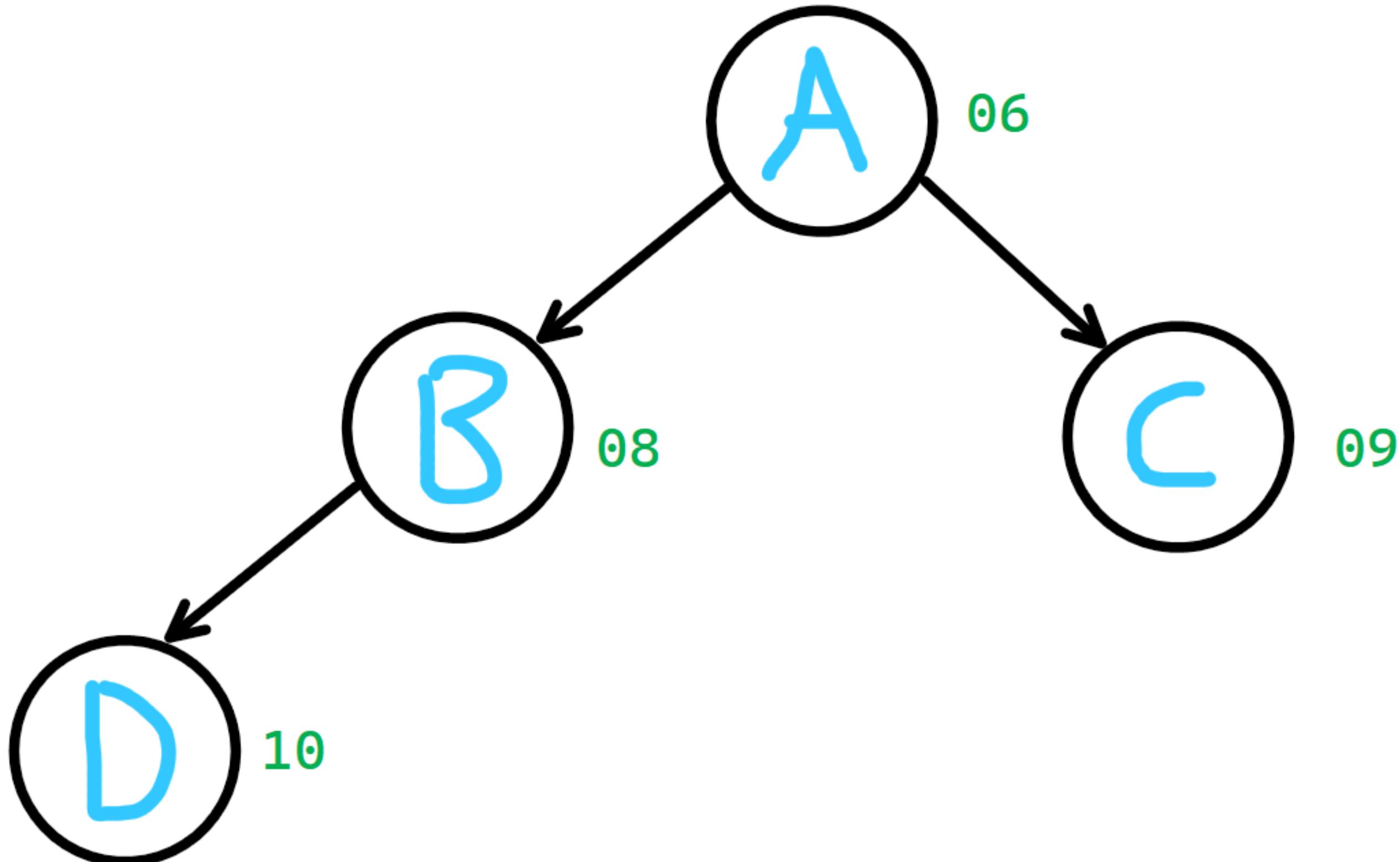
ÁRBOL DE PROCESOS

```
Soy el proceso A, mi PID es 2003306 y el PID de mi padre es 4080945
Soy el proceso B, mi PID es 2003308 y el PID de mi padre es 2003306
Soy el proceso C, mi PID es 2003309 y el PID de mi padre es 2003306
Soy el proceso D, mi PID es 2003310 y el PID de mi padre es 2003308
```

```
estudiante@NGEN211:~$ ps lf
```

F	UID	PID	PPID	PRI	NI	VSZ	RSS	WCHAN	STAT	TTY	TIME	COMMAND
0	1001	1979305	4080849	20	0	13676	5504	do_wai	Ss	pts/2	0:00	bash
4	1001	2003497	1979305	20	0	15204	3584	-	R+	pts/2	0:00	_ ps lf
0	1001	4080945	4080849	20	0	13808	4096	do_wai	Ss	pts/0	0:00	bash
0	1001	2003306	4080945	20	0	2776	1408	-	R+	pts/0	1:48	_ ./test
1	1001	2003308	2003306	20	0	2776	640	-	R+	pts/0	1:48	_ ./test
1	1001	2003310	2003308	20	0	2776	640	-	R+	pts/0	1:48	_ ./test
1	1001	2003309	2003306	20	0	2776	640	-	R+	pts/0	1:48	_ ./test
0	1001	1787614	1776272	20	0	8880	5376	do_sel	Ss+	pts/1	0:00	/usr/bin/bash --in

RESULTADOS





ANTES

```
Tasks: 406 total, 1 running, 405 sleeping, 0 stopped, 0 zombie
%Cpu0 : 1,0 us, 0,7 sy, 0,0 ni, 98,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 1,3 us, 1,0 sy, 0,0 ni, 97,3 id, 0,0 wa, 0,0 hi, 0,3 si, 0,0 st
%Cpu2 : 2,7 us, 0,7 sy, 0,0 ni, 96,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 1,3 us, 0,7 sy, 0,0 ni, 98,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

DESPUÉS

```
Tasks: 410 total, 6 running, 404 sleeping, 0 stopped, 0 zombie
%Cpu0 : 98,3 us, 0,3 sy, 0,0 ni, 1,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,0 us, 1,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

Conclusiones

Creación de procesos:

La función permite a los sistemas operativos crear procesos independientes de manera eficiente, promoviendo la multitarea, el paralelismo y la ejecución segura de aplicaciones.

Escalabilidad:

Los servidores y programas pueden manejar múltiples usuarios o solicitudes simultáneamente, garantizando rendimiento y disponibilidad.

Ejecución de programas externos:

La combinación de fork() y exec() es esencial para que los procesos lancen otros programas o comandos, siendo una parte fundamental para aplicaciones como las shells de línea de comandos.

Bases para sistemas operativos:

El estudio de la función ayuda a comprender conceptos esenciales como la jerarquía de procesos, el uso de recursos del sistema, la interacción entre procesos y la creación de aplicaciones robustas.

REFERENCIAS

- Patel, K. (2015, junio 16). Fork() in C. GeeksforGeeks. <https://www.geeksforgeeks.org/fork-system-call/>
- Como funciona la función fork(). (s/f). Stack Overflow en español. Recuperado el 31 de marzo de 2025, de <https://es.stackoverflow.com/questions/179414/como-funciona-la-funci%C3%B3n-fork>
- Z/OS. (2023, abril 28). Ibm.com. <https://www.ibm.com/docs/en/zos/2.5.0?topic=functions-fork-create-new-process>
- Improve, G. (2017, julio 24). Difference between fork() and exec(). GeeksforGeeks. <https://www.geeksforgeeks.org/difference-fork-exec/>
- HaiyingYu. (2024, 14 diciembre). Condiciones de carrera y interbloqueos - Visual Basic. Microsoft Learn. <https://learn.microsoft.com/es-es/troubleshoot/developer/visualstudio/visual-basic/language-compilers/race-conditions-deadlocks>
- AIX 7.1. (s. f.). <https://www.ibm.com/docs/es/aix/7.1?topic=processes->
- Raj, A. (2022, 10 marzo). Zombie and Orphan Process in OS | Scaler Topics. Scaler Topics. <https://www.scaler.com/topics/operating-system/zombie-and-orphan-process-in-os/>
- L. Nyman y M. Laakso, "Notes on the History of Fork and Join", IEEE Ann. Hist. Comput., vol. 38, n.º 3, pp. 84–87, julio de 2016. Accedido el 1 de abril de 2025. [En línea]. Disponible: <https://doi.org/10.1109/mahc.2016.34>
- "The fork() System Call". CS Lab webserver. Accedido el 1 de abril de 2025. [En línea]. Disponible: <https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html>
- V. Sharma. "Fork() System Call | Scaler Topics". Scaler Topics. Accedido el 1 de abril de 2025. [En línea]. Disponible: <https://www.scaler.com/topics/fork-system-call/>

Gracias por su atención





QUIZ

