



## Llamadas básicas

### Crear un proceso anidado

- ▼ La llamada a sistema `fork` se ejecuta de la siguiente forma

```
int fork();
```

#### Creación:

- El proceso que realiza esta llamada, crea un nuevo proceso que se jerarquiza como **su hijo**.
- Padre e hijo se ejecutarán de forma **concurrente**.
- El hijo parte con una **memoria idéntica** al padre (Código, Datos, etc)
- El hijo empieza a ejecutar en el punto en que se crea, ya que por el punto anterior,  $PC_{hijo} = PC_{padre}$  .
- Si tomamos el valor de retorno de fork, el padre y el hijo obtienen valores distintos:
  - El padre obtiene el PID del hijo.
  - El hijo tiene un 0.

#### Herencia:

### Terminar o esperar una ejecución

- ▼ Un proceso puede terminar su ejecución `exit(0)`

#### Terminar de forma involuntaria:

Un proceso puede ser matado, y terminar de forma involuntaria. Esto se consigue con Signals, que no tratamos en este apartado.

#### Terminación voluntaria:

Si un proceso debe terminar, tiene que tener la llamada `exit(0)` en su código

- ▼ También puede esperar a que termine un hijo `waitpid`

Para sincronizar al padre con la finalización del hijo, usamos `waitpid`.

- El proceso espera (e incluso bloquea si es necesario) hasta que termine un hijo cualquiera o uno en concreto:

### Mutación de un ejecutable

Cuando hacemos `fork`, el espacio de direcciones se pasa al hijo. Si queremos ejecutar un código distinto, tendremos que recurrir a **mutar** el proceso.

Como sabemos el id del archivo?

- ▼ Para mutar el proceso, tendremos que usar la llamada `execlp`

Con esta llamada a sistema, el proceso cambia su ejecutable por otro que le pasaremos por parámetro.

- Cambia todo el **contenido de memoria**
- Se mantiene la identidad del proceso
  - Signals pendientes, contadores, PID, etc.
- Se revierte cualquier cambio al tratamiento de los signals

#### Como se ejecuta

Dentro del hilo de ejecución del proceso que queremos mutar, deberemos añadir la siguiente línea de código:

```
execlp("exeName", "exeName", arg1,
```

El hijo heredará algunos valores del padre, y otros no.

#### **Hereda:**

- Memoria del padre
- Programación de los signals
- Usuario
- Variables de entorno

#### **No hereda:**

- PID del padre
- Contadores internos (Not useful)
- Alarmas y signals pendientes del padre

#### **Que termine un hijo cualquiera:**

- `waitpid(-1, NULL, 0)`

#### **Que termine un hijo en concreto:**

- `waitpid(PIDhijo, NULL, 0)`

Donde `"exeName"` es el nombre del archivo ejecutable al que queremos mutar, y `"arg1"` hasta `"argn"` son argumentos que necesite el ejecutable. Puede ir de 0 a  $n$ .

El hijo puede mandar al padre una señal de finalización mediante `exit(0)` y el padre la recoge mediante `waitpid`:

- Mientras el padre no la consulta, se quedará en **Zombie**. (Por eso conviene hacer un `waitpid`)