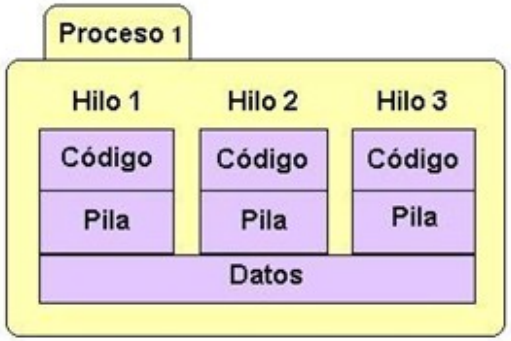
Jorge Hernández Ramírez

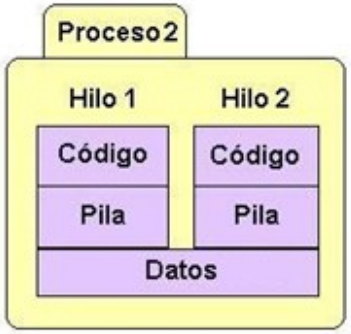
Indice

1. Procesos vs Hilos
2. Cuando se ejecutan los procesos
3. Fork
4. Clone
5. ¿Cómo se ejucuta el fork?
6. do_fork()
7. copy_process
8. Código

Procesos vs Hilos

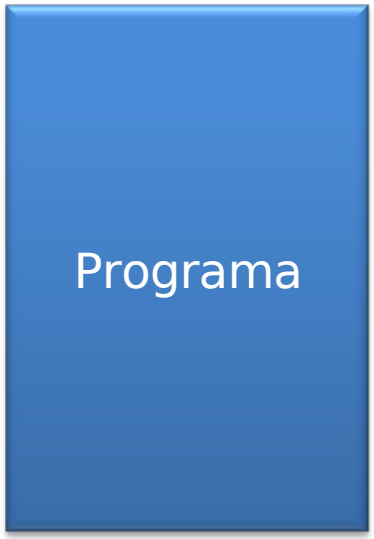
- ✓Proceso
  - ❑Manejados por el sistema operativo
  - ❑Me
    - o
    - o
    - o
- ✓Hilos
  - ❑Ma
  - ❑Memoria compartida entre hilos de un mismo proceso
    - oInstrucciones
    - oPila





¿Cuando se crean los procesos?

usuario



fork - clone

Fork

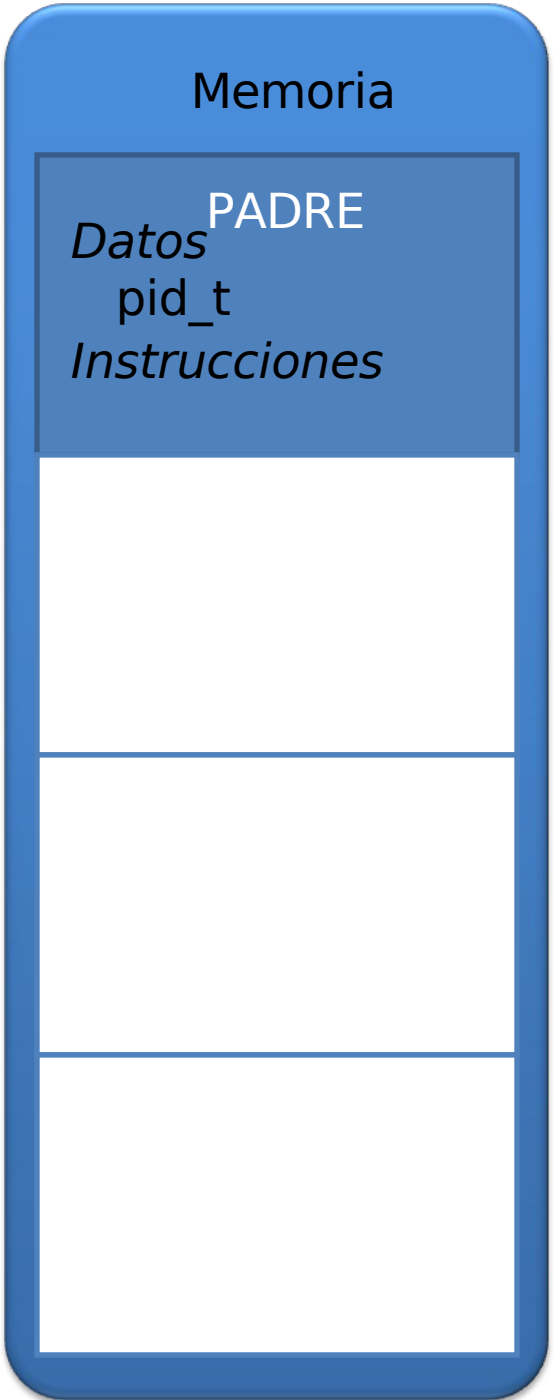
```
#include  <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid = fork();

    if(pid  == -1)
    {
        printf("Error al crear el proceso
hijo\n");
        return 0;
    }

    if(pid)
       printf("Soy el padre");
    else
       printf("Soy el hijo");

     return 0;
}
```



Fork

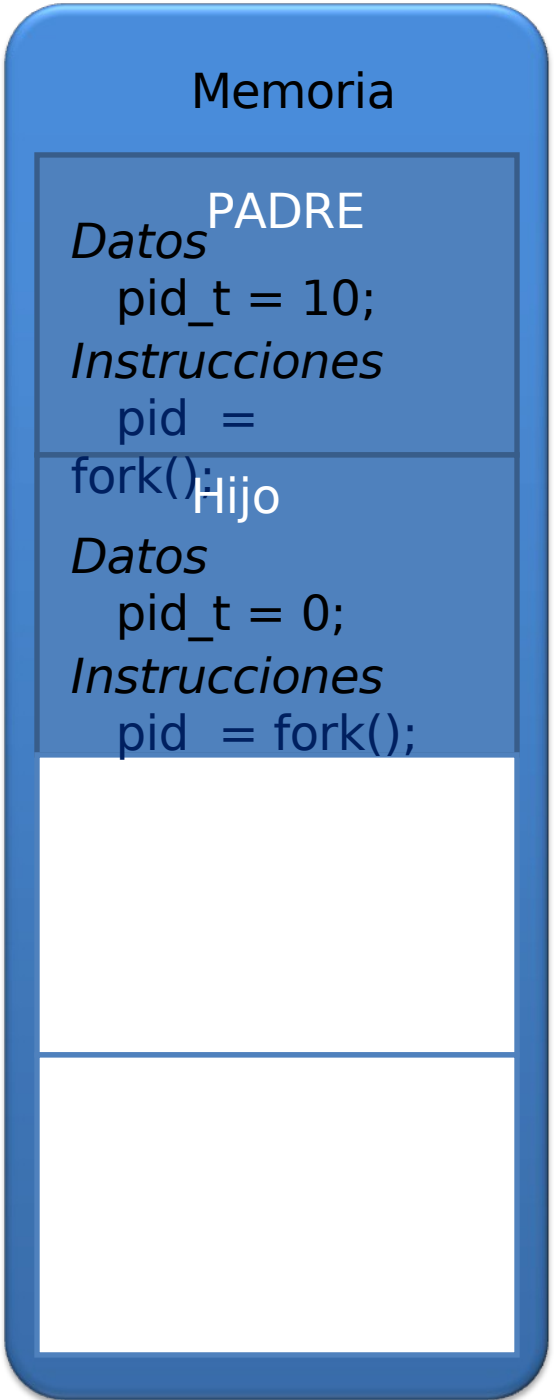
```
#include  <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid = fork();

    if(pid  == -1)
    {
        printf("Error al crear el proceso
hijo\n");
        return 0;
    }

    if(pid)
       printf("Soy el padre");
    else
       printf("Soy el hijo");

     return 0;
}
```



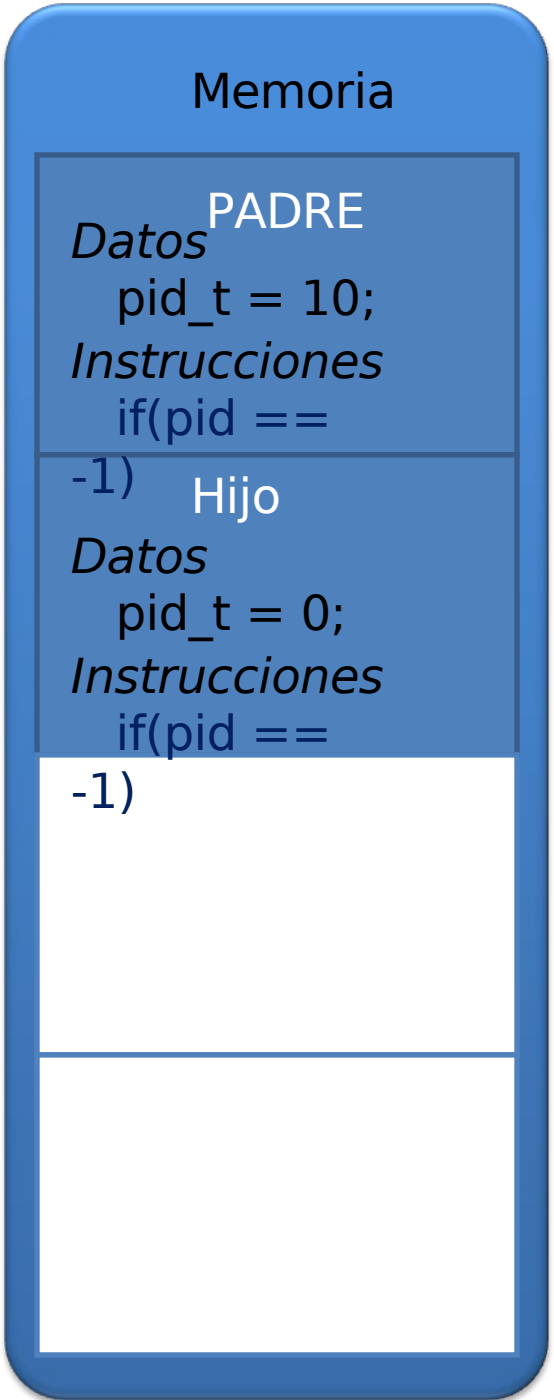
```
#include  <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid = fork();

     if(pid  == -1)
    {
        printf("Error al crear el proceso
hijo\n");
        return 0;
    }

    if(pid)
       printf("Soy el padre");
    else
       printf("Soy el hijo");

     return 0;
}
```



Fork

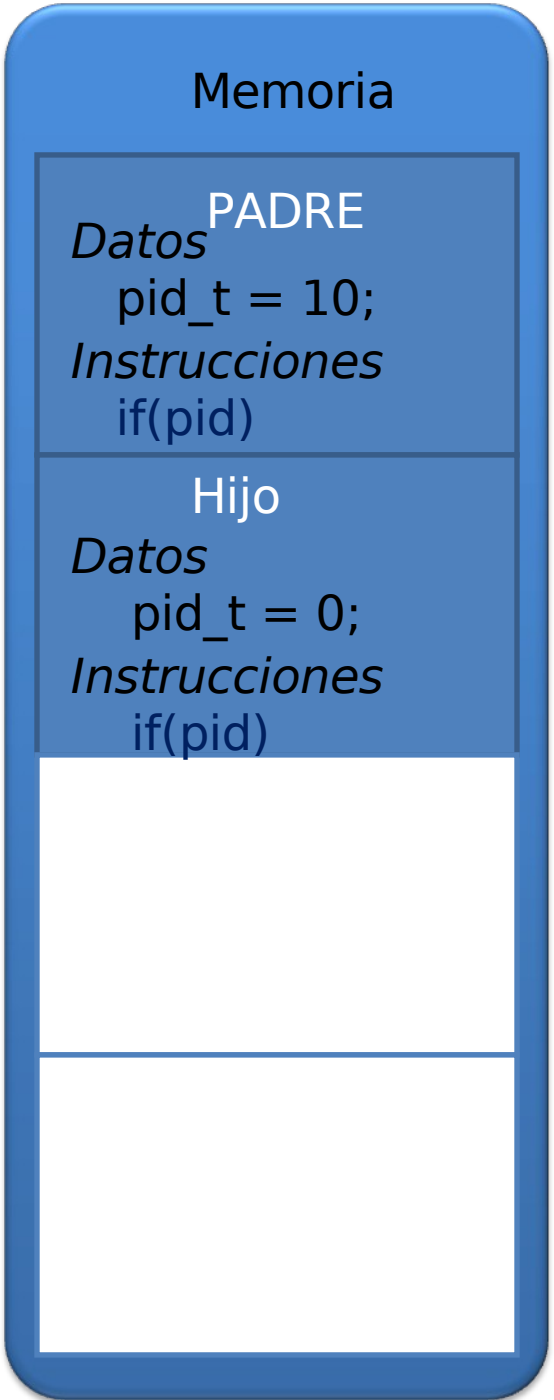
```
#include  <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid = fork();

    if(pid  == -1)
    {
        printf("Error al crear el proceso
hijo\n");
        return 0;
    }

    if(pid)
      printf("Soy el padre");
    else
      printf("Soy el hijo");

     return 0;
}
```

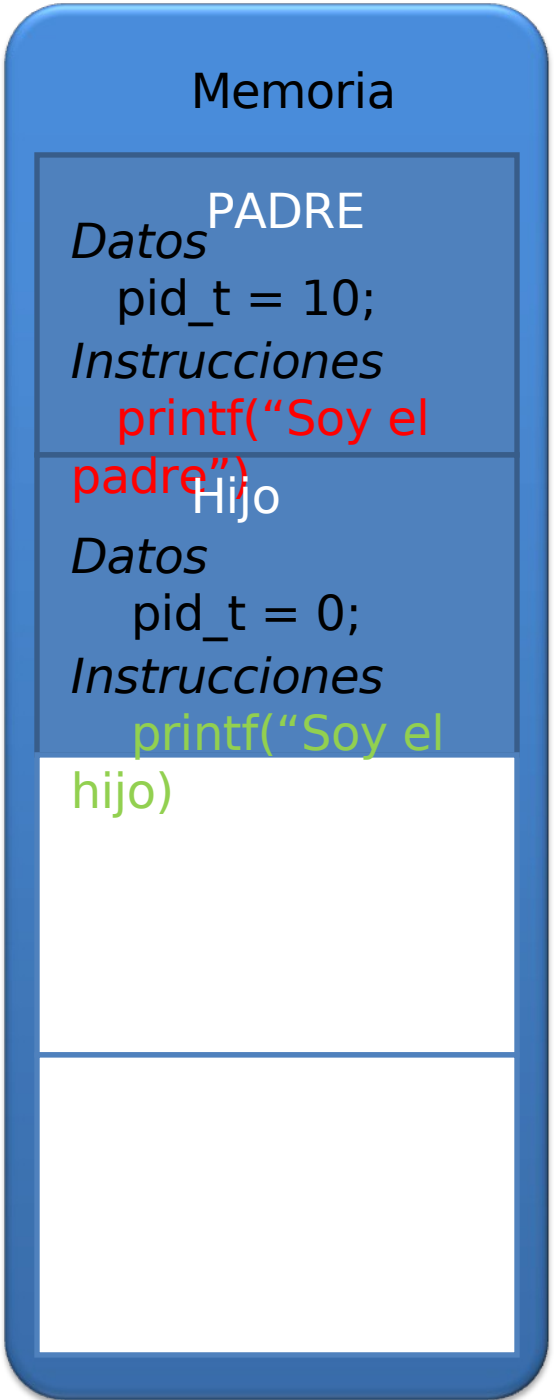


```
#include  <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid = fork();

    if(pid  == -1)
    {
        printf("Error al crear el proceso
hijo\n");
        return 0;
    }

    if(pid)
      printf("Soy el padre");
    else
      printf("Soy el hijo");
     return 0;
}
```



Fork

- ✓Cuando se hace un fork se crea un task_struct a partir del task_struct del proceso padre
- ✓Al hijo se le asigna PID propia
- ✓Al hijo se le copia
  - ❑Espacio de direccionamiento
  - ❑Descriptores de señales
  - ❑…
- ✓No se copia los descriptores de fichero, sino que se comparten.

Al hacer un fork, esta función nos devuelve

- ✓ -1 → entonces ha habido algún error y por tanto no se ha creado el hijo
- ✓ 0 → estamos en el proceso hijo
- ✓ otro → estamos en el proceso padre. El valor devuelto por el fork es el identificador del hijo creado.

Clone(I)

Igual que el fork pero permite especificar qué queremos que compartan padre e hijo. *Pueden compartir mismo PID*

- ✓Espacio de direccionamiento
- ✓Información de control del sistema de archivos (file system)
- ✓Descriptores de archivos abiertos.
- ✓Gestores de señales o PID.

Clone(flags)

Nombre Flag	Descripción
CLONE_VM	Si se pone CLONE_VM, los procesos padre e hijo se ejecutan en el mismo espacio de memoria (por lo que una modificación en un proceso se refleja también en el otro).
CLONE_FS	Si se pone CLONE_FS, los procesos padre e hijo comparten la misma información del sistema de ficheros. Ésta incluye la raíz del sistema de ficheros, el directorio de trabajo actual y el valor de umask.
CLONE_FILES	Si se pone CLONE_FILES, los procesos padre e hijo comparten la misma tabla de descriptores de fichero. Los descriptores de fichero siempre se refieren a los mismos ficheros en el padre y en el proceso hijo.
CLONE_SIGHAND	Si se pone CLONE_SIGHAND, los procesos padre e hijo comparten la misma tabla de manejadores de señal.
CLONE_PTRACE	Si el proceso padre está siendo depurado, el proceso hijo también estará siendo depurado.
CLONE_VFORK	Utilizada por vfork para indicar que despierte al padre al morir.
CLONE_PARENT	Si CLONE_PARENT está presente, el padre del nuevo hijo será el mismo que el del proceso invocador. Si CLONE_PARENT no está presente, el padre del hijo es el proceso invocador.
CLONE_THREAD	Si CLONE_THREAD está presente, el proceso hijo se pone en el mismo grupo de hilos que el proceso invocador y fuerza al hijo a compartir el descriptor de señal con el padre.

Clone(flags)

CLONE_NEWNS	Esta activa si la copia necesita su propio namespace, es decir, su propia perspectiva de los ficheros montados.
CLONE_SYSVSEM	Padre e hijo comparten una única lista de valores de deshacer para un semáforo de tipo System V.
CLONE_SETTLS	Crea un nuevo segmento de almacenamiento local para un hilo.
CLONE_PARENT_SETTID	Almacena el PID del hijo en la variable parent_tidptr tanto en la zona de memoria del padre como en la del hijo.
CLONE_CHILD_CLEARTID	Elimina el PID del hijo de “child_tidptr del espacio de memoria en caso de que éste exista. Además despierta el “futex” en esa dirección.
CLONE_UNTRACED	Usada por el núcleo para deshabilitar CLONE_PTRACE.
CLONE_CHILD_SETTID	Escribe el PID del hijo en “child_tidptr” en el espacio de memoria del hijo
CLONE_STOPPED	Fuerza al hijo a comenzar en estado TASK_STOPPED.

¿Cómo se ejecuta el fork?

- ✓La función fork de la librería libc coloca los parámetros de la llamada en los registros del procesador y se ejecuta la instrucción INT 0x80.

- ✓Se conmuta a modo núcleo y, mediante las tablas IDT y GDT, se llama a la función sys_call.

- ✓La función sys_call busca en la sys_call_table la dirección de la llamada al sistema sys_fork.

- ✓La función sys_fork llama a la función do_fork, que es la función principal del archivo fork.c.

¿Cómo se ejecuta el fork?

Procesador registros
Instrucción Pid_t=fork()

Modo usuario

Memoria

Proceso de usuario
Datos
pid_t
Instrucciones
fork()

libc
fork()
//inicializa reg
//INT 0x80

sys_call
//llama a sys_fork

sys_fork
//llama a do_fork

¿Cómo se ejecuta el fork?

Procesador registros
Instrucción //inicial. reg

Modo usuario

Memoria

Proceso de usuario

Datos
pid_t
Instrucciones
fork()

libc
fork()
//inicializa reg
//INT 0x80

sys_call
//llama a sys_fork

sys_fork
//llama a do_fork

¿Cómo se ejecuta el fork?

Procesador registros
Instrucción //INT 0x80

Modo usuario

Memoria

Proceso de usuario

Datos
pid_t

Instrucciones
fork()

libc
fork()
//inicializa reg
//INT 0x80

sys_call
//llama a sys_fork

sys_fork
//llama a do_fork

¿Cómo se ejecuta el fork?

Procesador registros
Instrucción //llama a sys_fork

Modo nucleo

Memoria

Proceso de usuario
Datos
pid_t
Instrucciones
fork()

libc
fork()
//inicializa reg
//INT 0x80

sys_call
//llama a sys_fork

sys_fork
//llama a do_fork

¿Cómo se ejecuta el fork?

Procesador registros
Instrucción //llama a do_fork

Modo nucleo

Memoria

Proceso de usuario
Datos
pid_t
Instrucciones
fork()

libc
fork()
//inicializa reg
//INT 0x80

sys_call
//llama a sys_fork

sys_fork
//llama a do_fork

Problemas (fork - clone)

En la función copy_process a la variable retval se le pueden asignar dos códigos de error en función de los errores que puediesen surgir.

eagain

eanomem

do_fork()

- ✓ Implementa la llamada al sistema fork() y clone()
- ✓ Cosas que hace
  - ❑ Llamar a la función copy_process()
  - ❑ Asignar PID al hijo
  - ❑ Devolver PID

copy_process()

- ✓ Función que devuelve un puntero al task_struct creado.

Nota: Todo proceso en Linux debe tener un task_struct asociado, donde se guarda toda la información del proceso.

- ✓ Cosas que hace
  - ❑ Crea un task_struct meidante la función dup_task_struct
  - ❑ Inicializa los campos del task_struct, principalmente mediante las funciones copy_fs, copy_sighand, copy_signal, copy_mm, copy_namespace
  - ❑ Devolver puntero al task_struct creado

Nota: Los flags que se le pasan a la función copy_process le indican que debe copiar o que compartir

En caso del fork estos flags no existirán por lo que se hará una copia exacta