

CICLO FORMATIVO DE GRADO SUPERIOR

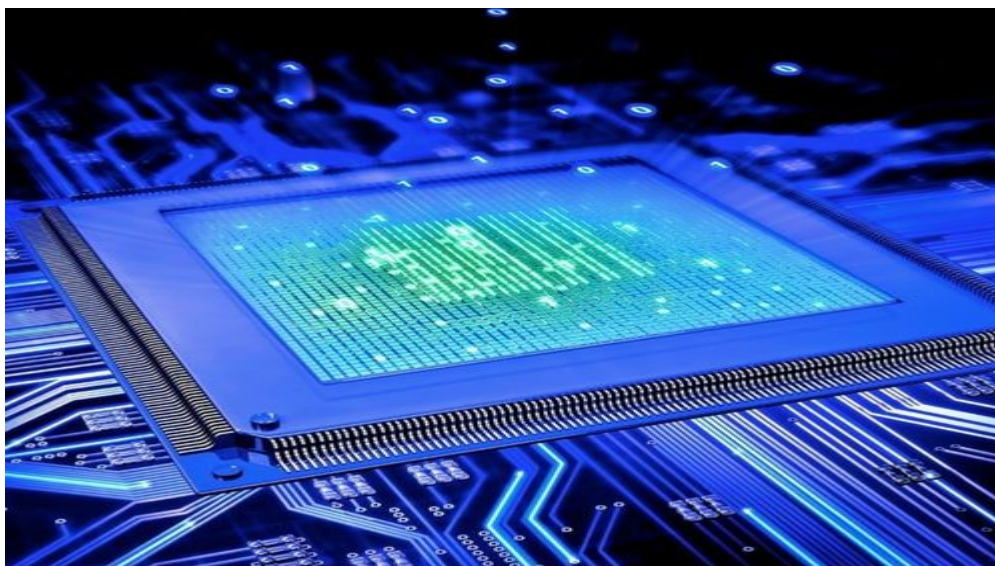
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

CURSO 2024/2025

MÓDULO: PROGRAMACIÓN DE SERVICIOS Y PROCESOS

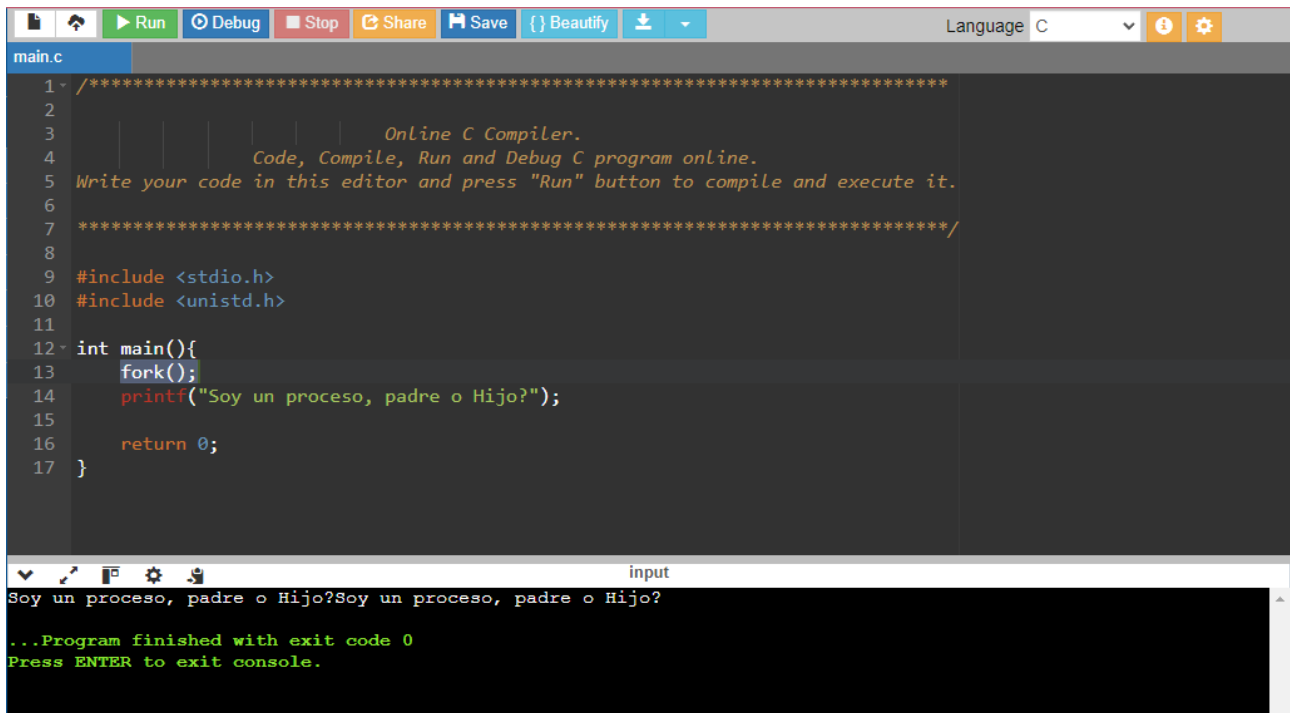
Práctica 1 : Unidad 1 → Ejemplos sencillos fork()

Rafael Álvaro Palomares Linares



1. Elaborar un documento pdf, con capturas de pantalla de la ejecución de cada uno de los ejemplos del fork. Debéis explicar claramente qué es lo que hacen.

Ejemplo 1



```
main.c
1  /*****
2
3      Online C Compiler.
4      Code, Compile, Run and Debug C program online.
5      Write your code in this editor and press "Run" button to compile and execute it.
6
7      *****/
8
9  #include <stdio.h>
10 #include <unistd.h>
11
12 int main(){
13     fork();
14     printf("Soy un proceso, padre o Hijo?");
15
16     return 0;
17 }
```

input

Soy un proceso, padre o Hijo?Soy un proceso, padre o Hijo?

...Program finished with exit code 0
Press ENTER to exit console.

El fork() hace una llamada al sistema y crea una copia del proceso en ejecución llamándose esta proceso hijo y la original proceso padre. Después de la llamada se siguen ejecutando ambos procesos simultáneamente sin seguir un orden específico. Cuando ambos procesos lleguen a “return 0;” terminarán su ejecución.

Ejemplo 2

```

main.c
5 Write your code in this editor and press "Run" button to compile and execute it.
6
7 *****/
8
9 #include <stdio.h>
10 #include <unistd.h>
11
12 int main()
13 {
14     pid_t pid;
15     fork();
16
17     switch(pid) {
18         case 0:
19             printf("Soy el padre, este es mi pid %d y este el de mi hijo %d\n", getpid(), getppid());
20             break;
21         default:
22             printf("Soy el hijo, este es mi pid %d y este el de mi padre %d\n", getppid(), getpid());
23             break;
24     }
25
26     fork();
27
28     switch(pid) {
29         case 0:
30             printf("Soy el padre, este es mi pid %d y este el de mi hijo %d\n", getpid(), getppid());
31             break;
32         default:
33             printf("Soy el hijo, este es mi pid %d y este el de mi padre %d\n", getppid(), getpid());
34             break;
35     }
36
37     fork();
38
39     switch(pid) {
40         case 0:
41             printf("Soy el padre, este es mi pid %d y este el de mi hijo %d\n", getpid(), getppid());
42             break;
43         default:
44             printf("Soy el hijo, este es mi pid %d y este el de mi padre %d\n", getppid(), getpid());
45             break;
46     }
47
48     return 0;
49 }

```

input

```

Soy el padre, este es mi pid 3332 y este el de mi hijo 3331
Soy el padre, este es mi pid 3336 y este el de mi hijo 3332
Soy el padre, este es mi pid 3332 y este el de mi hijo 3331
Soy el padre, este es mi pid 3336 y este el de mi hijo 3332
Soy el padre, este es mi pid 3337 y este el de mi hijo 3332
Soy el padre, este es mi pid 3332 y este el de mi hijo 3331
Soy el padre, este es mi pid 3338 y este el de mi hijo 3336
Soy el padre, este es mi pid 3336 y este el de mi hijo 3332
Soy el padre, este es mi pid 3339 y este el de mi hijo 3332
Soy el padre, este es mi pid 3337 y este el de mi hijo 3332

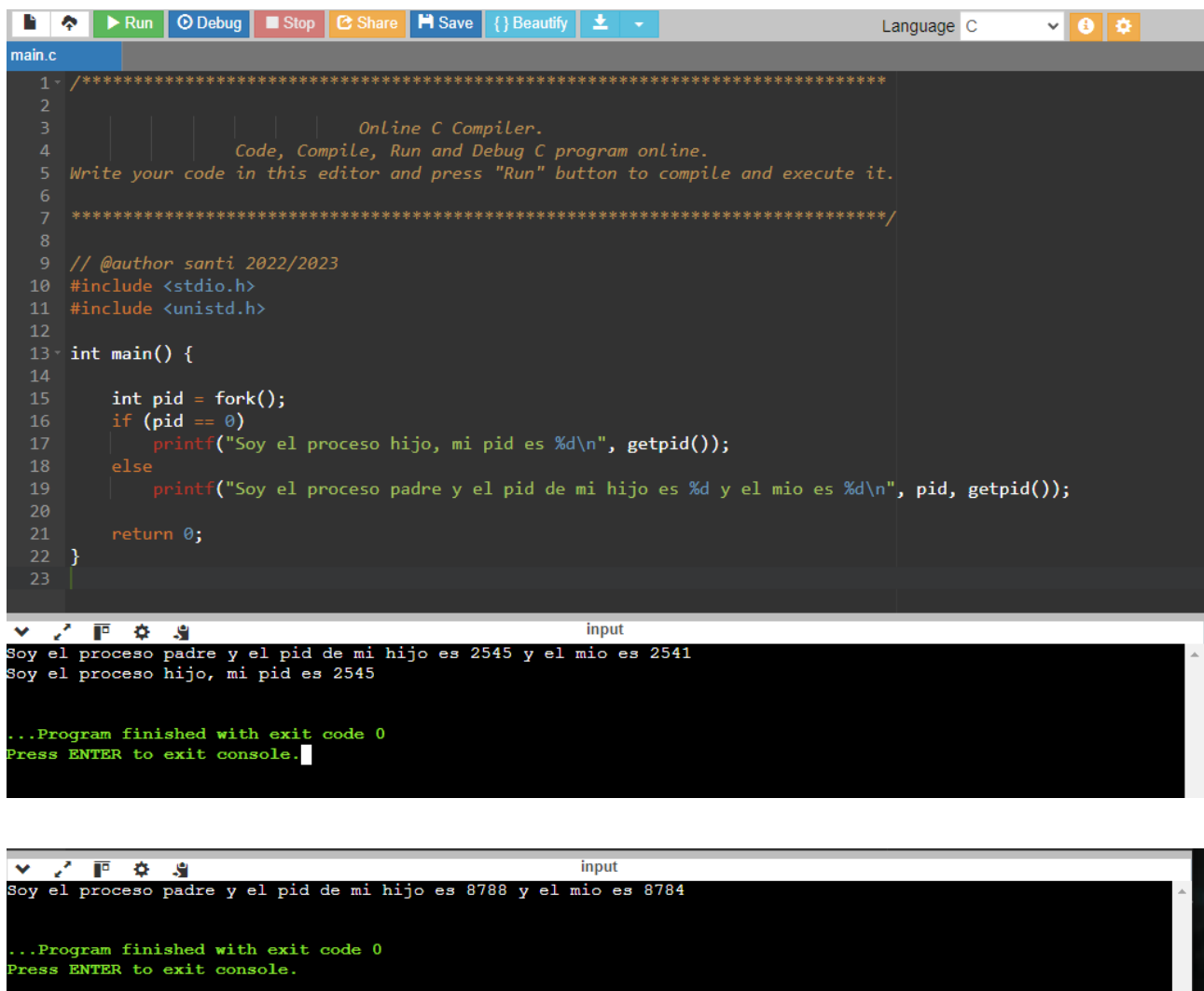
```

En este ejemplo el primer fork crea una copia del proceso original siendo, por ejemplo el padre **p0** y el hijo **p1**, el proceso es hasta el siguiente fork por lo cual se muestran las dos primeras salidas.

Una vez se llega al segundo fork se crea una copia de **p0** y **p1** siendo **p2** hijo de **p0** y **p3** hijo de **p2** (por ejemplo). Por lo cual se accederá al segundo switch mostrando por pantalla otras cuatro salidas. En este punto tenemos un total de 2 elevado a 2 procesos.

En el siguiente fork se creará una copia de los 4 procesos en ejecución siendo, por ejemplo, **p5** hijo de **p0**, **p6** hijo de **p1**, **p7** hijo de **p2** y **p8** hijo de **p3**. Después se ejecutará el siguiente bloque switch, otras 4 salidas por cada nueva copia creada mostrando un total de 10 salidas. A todo esto el pid es el identificados asociado a cada proceso.

Ejemplo 3



```

1  /*****
2
3      Online C Compiler.
4      Code, Compile, Run and Debug C program online.
5      Write your code in this editor and press "Run" button to compile and execute it.
6
7      *****/
8
9  // @author santi 2022/2023
10 #include <stdio.h>
11 #include <unistd.h>
12
13 int main() {
14
15     int pid = fork();
16     if (pid == 0)
17         printf("Soy el proceso hijo, mi pid es %d\n", getpid());
18     else
19         printf("Soy el proceso padre y el pid de mi hijo es %d y el mio es %d\n", pid, getpid());
20
21     return 0;
22 }
23

```

input

```

Soy el proceso padre y el pid de mi hijo es 2545 y el mio es 2541
Soy el proceso hijo, mi pid es 2545

...Program finished with exit code 0
Press ENTER to exit console.

```

input

```

Soy el proceso padre y el pid de mi hijo es 8788 y el mio es 8784

...Program finished with exit code 0
Press ENTER to exit console.

```

En este código se crea una copia del proceso padre y se le asigna el pid de este proceso a una variable llamada pid. Luego el código se sigue ejecutando de tal manera que si el pid es igual a 0 significaría que estamos en el proceso hijo, si es diferente a 0 estamos en el proceso padre.

Se dio un caso en el que solo apareció la salida que indica que estamos en el proceso padre. Esto se debe a que en algún momento ocurrió un error durante la creación del proceso hijo.

Ejemplo 4

```

1  /*****
2
3      Online C Compiler.
4      Code, Compile, Run and Debug C program online.
5      Write your code in this editor and press "Run" button to compile and execute it.
6
7  *****/
8
9  #include <stdio.h>
10 #include <unistd.h>
11
12 int main()
13 {
14     int conta = 1;
15     if(fork()){
16         for(int i = 0; i < 5000000; i++)
17             conta = conta;
18         printf("Soy el proceso padre y voy a incrementar la variable conta = %d\n", ++conta);
19     } else {
20         for(int i = 0; i < 50000; i++)
21             conta = conta;
22         printf("Soy el proceso hijo y voy a decrementar la variable conta = %d\n", --conta);
23     }
24
25     return 0;
26 }

```

input

```

Soy el proceso hijo y voy a decrementar la variable conta = 0
Soy el proceso padre y voy a incrementar la variable conta = 2

...Program finished with exit code 0
Press ENTER to exit console.

```

En este proceso se crea una copia del proceso padre en el que si fork es 0 estaríamos en el proceso hijo y si es distinto de 0 estaríamos en el proceso padre.

Luego se ejecuta un bucle que realiza más iteraciones en el proceso padre que en el proceso hijo. Esto significa que el proceso hijo va a terminar su ejecución antes que el padre por lo cual la salida del proceso hijo se muestra antes en pantalla.

Ejemplo 5

```

7  *****/
8
9  // @author santi 2022/2023
10 #include <stdio.h>
11 #include <unistd.h> // llamada del sistema fork
12 #include <sys/types.h>
13 #include <sys/wait.h> // define tipos
14 #include <stdio.h> // llamada E/S
15 #include <stdlib.h> // llamada exit
16
17 int main() {
18     int op = 1, estado;
19     pid_t pid;
20
21     pid = fork();
22     switch (pid) {
23         case -1:
24             printf("Ha pasado algo y error inesperado\n");
25             exit(-1);
26         case 0:
27             printf("Soy el hijo y mi pid es %d\n", getpid());
28             break;
29         default:
30             printf("Soy el padre con pid %d y siempre me esperaré a que finalice primero mi hijo con pid %d\n", pid, estado);
31             wait(&estado);
32     }
33 }
34

```

input

```

Soy el padre con pid 19035 y siempre me esperaré a que finalice primero mi hijo con pid 19039
Soy el hijo y mi pid es 19039
...Program finished with exit code 0
Press ENTER to exit console.

```

Se crea una copia del proceso padre. En el switch se evalúan los posibles casos, uno de ellos es -1 que aparece cuando se produce un error. En caso de que no ocurra ningún error los procesos continuarán su ejecución normal con la diferencia de que si el proceso padre se ejecuta primero este siempre esperará a que se termine la ejecución de su hijo para continuar. Esto lo hace la función wait().

Ejemplo 6

```

main.c
6
7 *****/
8
9 // @author santi 2022/2023
10 #include <stdio.h>
11 #include <sys/types.h>
12 #include <sys/wait.h>
13 #include <unistd.h>
14 #include <stdlib.h>
15
16 void pruebaParametros(int args, char ** argv){
17     printf("El numero de argumentos son %d\n", args);
18     for(int i=0; i<args; i++){
19         printf("Arg(%d) = %s\n", i, argv[i]);
20     }
21 }
22
23 int main(int args, char **argv) {
24     pid_t pid;
25     int estado;
26     pid = fork();
27
28     switch (pid) {
29         case -1:
30             printf("Error inesperado\n");
31             exit(-1);
32         case 0:
33             printf("Soy el proceso hijo y voy a ejecutar el cat con pid %d, \n", getpid());
34             if( execvp(argv[1], & argv[0]) < 0){
35                 printf("Error de cojones");
36                 exit(-2);
37             }
38             printf("Jaaaa, ya he ejecutado el comando pasado y sigo siendo %d, ¿Que ha pasado?", getpid());
39             break;
40         default:
41             printf("Soy el padre con pid %d y voy a esperar que finalice primero mi hijo con pid %d\n", getpid(), pid);
42             wait(&estado);
43             break;
44     }
45 }
46
input
Soy el padre con pid 10053 y voy a esperar que finalice primero mi hijo con pid 10057
Soy el proceso hijo y voy a ejecutar el cat con pid 10057,
...Program finished with exit code 0
Press ENTER to exit console.

```

Crea una copia del proceso y se comprueba si se está ejecutando el proceso padre o el hijo. Si ocurre un error se mandará un mensaje de error. Si el proceso padre se ejecuta antes siempre esperará al proceso hijo con la variable que almacena su estado “**estado**” en la instrucción wait para continuar.