

## Actividades de laboratorio

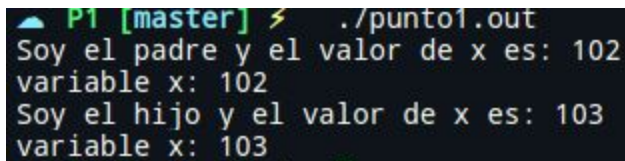
1. Escriba un programa que llame un `fork()`. Antes del llamado del `fork()`, declare una variable de acceso (por ejemplo, `x`) y asígnele un valor (por ejemplo, 100). Responda las siguientes preguntas:

- ¿Cuál es el valor de la variable en el proceso hijo?

R/ El valor de la variable es 102, tomando como valor inicial de `x = 100` y sumándole en el proceso hijo 2 unidades.

- ¿Qué sucede con la variable cuando el proceso hijo y el padre cambian el valor de `x`?

R/ La variable `x` conserva el valor que cada proceso le da, ya que pueden tener las mismas variables, sin embargo estas son independientes, pues están alojadas para cada proceso en diferentes espacios de memoria.

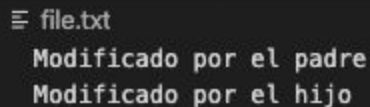


```
P1 [master] $ ./punto1.out
Soy el padre y el valor de x es: 102
variable x: 102
Soy el hijo y el valor de x es: 103
variable x: 103
```

2. Escriba un programa que abra un archivo (con la llamada `open()`) y entonces llame a `fork()`. Nota: El siguiente [enlace](#) puede ser de utilidad para entender la llamada `open()`.

- ¿Pueden el padre y el hijo acceder al file descriptor retornado por `open()`?

R/ Sí, los dos procesos tienen el mismo descriptor que pueden modificar el archivo.



```
file.txt
Modificado por el padre
Modificado por el hijo
```

- ¿Qué pasa si ellos empiezan a escribir el archivo de manera concurrente, es decir, a la misma vez?

R/ Acontece que con la concurrencia, se cambia el orden de escritura, no se va a poder observar un orden secuencial sino según como fue planificado en el scheduler.

3. Escriba un programa usando `fork()`. El proceso hijo imprimirá "Hello"; el proceso padre imprimirá "goodbye". Usted deberá asegurar que el proceso hijo imprima en primer lugar; ¿usted podría hacer esto sin llamar `wait()` en el padre?

```
if (pid)
{ /* padre */
    sleep(1);
    printf("goodbye\n");
}
else
{ /* hijo */
    printf("hello\n");
}
```

Si, utilizando la función `sleep`, podemos hacer que el padre espere la respuesta del hijo y obtener lo siguiente:

```
P3 [master] ./punto3.out
hello
goodbye
```

4. Escriba un programa que llame `fork()` y entonces llame alguna forma de `exec()` para correr el programa `/bin/ls`. Intente probar todas las variaciones de la familia de funciones `exec()` incluyendo (en linux) `execl()`, `execle()`, `execlp()`, `execv()`, `execvp()` y `execvpe()`. ¿Por qué piensa usted que existen tantas variaciones para la misma llamada básica?

```
MacBook-Air-de-Deiry:P4 deiry$ ./punto4
total 40
-rwxr-xr-x  1 deiry  staff   13K Mar 28 19:18 punto4
-rw-r--r--  1 deiry  staff  1.3K Mar 28 19:19 punto4.c
MacBook-Air-de-Deiry:P4 deiry$
```

R/ La función `exec()` tiene una familia de funciones que varían por sus diferentes parámetros, algunas funciones proporcionan una matriz de parámetros, mientras que otros consiste en una lista de valores y también de una matriz de su entorno.

6. Escriba ahora un programa que use `wait()` para esperar que el proceso hijo finalice su ejecución.

¿Cuál es el valor de retorno de la función `wait()`?

R/: El valor de retorno de la función `wait` en el padre fué de 24377, lo que equivale al pid del hijo, es decir al process ID.

¿Qué pasa si usted usa la función `wait` en el hijo?

R/: Cuando se usa la función `wait` en el hijo, el valor que arroja es de -1, lo que significa que no está esperando por ningún proceso hijo.

```
P5 [master] $ ./punto5.out
Hola soy el hijo y este es el valor de wait -1:
Hola soy el padre y este es el valor de wait 24377:
```

7. Haga un programa, como el del ejercicio anterior, con una breve modificación, la cual consiste en usar `waitpid()` en lugar de `wait()`. ¿Cuándo podría ser `waitpid()` útil?

Se podría usar `waitpid` cuando se tienen varios procesos hijos y se desea esperar por uno en específico, por lo que se le pasa el pid como argumento a la función `waitpid()`, comportamiento diferente al `wait()` porque esta función lo que hace es esperar la finalización de cualquier hijo.

```
P6 [master] $ ./punto6.out
Hola soy el hijo y este es el valor de waitpid -1:
Hola soy el padre y este es el valor de waitpid 30285:
```

Sin embargo con `waitpid` se puede obtener los mismos resultados del ejercicio anterior.

8. Escriba un programa que cree un proceso hijo y entonces en el proceso hijo cierre la salida estándar (`STDOUT_FILENO`). ¿Qué pasa si el hijo llama `printf()` para imprimir alguna salida después de cerrar el descriptor?

Cuando el hijo llama `printf()` para imprimir alguna salida después de cerrar el descriptor, este no retorna nada, no tiene en cuenta los llamados para escribir después de cerrado el descriptor.

```

int main(int argc, char *argv[])
{
    //int status;
    pid_t pid;
    pid = fork();
    if(pid == -1){
        printf("Error al crear proceso hijo");
    }
    if (pid)
    { /* padre */
        printf("Hola soy el padre\n");
    }
    else
    { /* hijo */
        printf("Hola soy el hijo antes de cerrar STDOUT\n");
        close(STDOUT_FILENO);
        printf("Hola soy el hijo después de cerrar STDOUT\n");
    }
    return 0;
}

```

Y esta es la salida:

```

P7 [master] ✂ ./punto7.out
Hola soy el hijo antes de cerrar STDOUT
Hola soy el padre

```

9. Escriba un programa que cree dos hijos y conecte la salida estándar de un hijo a la entrada estándar del otro usando la llamada a sistema `pipe()`.

```

MacBook-Air-de-Deiry:P8 deiry$ gcc -o punto8.out punto8.c
MacBook-Air-de-Deiry:P8 deiry$ ./punto8.out
Tamaño de bytes: 26
Mensaje de prueba pipe()#1
MacBook-Air-de-Deiry:P8 deiry$ █

```

Realizado por Estefany Muriel y Sofía Navas