 # SOLUCION LABORATORIO 1 - PROCESOS #
Por : ArcangeL Marin
Juan Carlos Londoño

Realizar los siguientes ejercicios:

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 x en el proceso hijo es el mismo x =100.

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

R/ Se va modificando A medida que el cada proceso lo va modificando. Ya que las variables están en direcciones diferentes según cada proceso.

2. Escriba un programa que abra un archivo (con la llamada ```open()```) y entonces llame a ```fork()```. **Nota: El siguiente [enlace](https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/) puede ser de utilidad para entender la llamada ```open()```.**

*** ¿Pueden el padre y el hijo acceder al file descriptor retornado por ```open()```?**
R/ Si ambos pueden acceder al file descriptor.

*** ¿Qué pasa si ellos empiezan a escribir el archivo de manera concurrente, es decir, a la misma vez?**
R/ Primero se escribe lo del proceso que ingrese al archivo, después se va escribiendo lo que cada proceso va haciendo.

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?

R/ Si es posible hacerlo haciendo uso de la función sleep(), de tal manera que en el proceso padre se ponga en espera un determinado tiempo mientras tanto el proceso hijo se ejecuta. Sin EMBARGO surge la inquietud de saber si realmente el proceso hijo demora un tiempo menor para garantizar que efectivamente
este se ejecute primero. Por lo tanto, consideramos poco confiable usar el sleep() como método para ejecutar un proceso des pues de otro.

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?

R/ Para poder dar o no información de las algunas variables con que se cuenta, por ejemplo la variable **PATH.** En este caso tanto execl como execv ignoran la variable PATH, que contiene las rutas de búsqueda. Para tener en cuenta esta variable pueden usarse las versiones execlp o execvp.

5. 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()```?, ¿Qué pasa si usted usa la función ```wait``` en el hijo?

R/ El valor de retorno fue 0 (cero),quiere decir que el proceso hijo al cual esperaba el padre finalizó.
Si se usa la función wait en el hijo el valor de retorno hubiese sido -1.

6. 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?

R/ Waitpid() es útil cuando se tienen varios hijos y deseamos que el proceso padre continúa ejecutándose cuando específicamente
el proceso hijo que él desea esperar termine.

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

R/ Si cerramos la salida STANDAR de algun programa, este no podra mostrar ninguna información en esta salida.

8. 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()```