**Sistemas Operativos**

*Prácticas curso 2021-2022*

Programación con llamadas al sistema: Práctica 3

1. Ejecuta la Version2 del intérprete de comandos. A continuación, ejecuta el comando “**ps lxw**” para ver tus procesos en ejecución. ¿Cuál de los procesos es el intérprete de comandos del sistema (el que se te proporciona al iniciar sesión? ¿Qué relación hay entre el intérprete de comandos del sistema y nuestro intérprete? ¿Y qué relación hay entre nuestro intérprete de comandos y el comando **ps** que acabas de ejecutar? Explica por qué existen dichas relaciones.

El intérprete de comandos del sistema es bash. Bash es el proceso padre de nuestro intérprete, puesto que lo estamos ejecutando mediante él. A su vez, el comando “ps lwx” es un proceso hijo del nuestro intérprete. Todo esto tiene sentido porque para ejecutar un “comando”, ambos intérpretes crean un hijo mediante “fork” y luego ejecutan un proceso diferente mediante “exec”.

1. Al realizar el ejercicio anterior observarías que tras la ejecución del comando **ps** no salió el *prompt* (el símbolo $) a continuación de su salida en pantalla. Si observas con detenimiento la pantalla verás que el *prompt* se imprimió entre el texto volcado por el comando **ps**. ¿Por qué ocurre esto?

Porque el proceso padre, el que no ejecuta los comandos del intérprete, no espera a que el hijo termine de ejecutar el comando introducido, que casi siempre tardará más en ejecutarse que imprimir un simple “$” por pantalla.

1. Sin salir de la Version2 del intérprete de comandos (si ya has salido vuelve a ejecutarlo), ejecuta el comando “ls l”. Observa como de nuevo no sale el *prompt* donde debería. A continuación sal de nuestro intérprete de comandos y ejecuta en el intérprete del sistema (el **bash**), el comando “**ls –l &**”. ¿Qué observas? Averigua qué significa poner un & al final de los comandos en UNIX. ¿Cómo se ejecutan entonces todos los comandos en la Version2 de nuestro intérprete?

Ocurre lo mismo que ocurre en la versión dos de nuestro intérprete, puesto que no se espera a que termine el comando introducido para poner el prompt de nuevo. La función de “&” es esta precisamente, que el comando se ejecute *en segundo plano* y que por lo tanto se pueda seguir ejecutando cualquier otra cosa en primer plano.

1. Ejecuta la Version2 del intérprete de comandos. Si ya está en ejecución sal y vuelve a ejecutarlo (es MUY IMPORTANTE ejecutar todo desde el comienzo). Ejecuta (todos seguidos) los siguientes comandos: ls –l clear sleep 2 ps lxw

Examina la salida del último comando (comando **ps**). ¿Qué observas? ¿En qué estado están los procesos anteriormente ejecutados?

Los comandos anteriores aparecen en estado “defunct”.

1. Ejecuta de nuevo la Version2. Si ya está en ejecución sal y vuelve a ejecutarlo (es MUY IMPORTANTE ejecutar todo desde el comienzo). Ejecuta un comando que no exista (por ejemplo ***novale***) y a continuación haz un “**ps lxw**”. Observa como parece que hay ahora dos procesos que son la Version2 del intérprete cuando solo debería haber uno. Ejecuta ahora otra vez un comando que no exista y a continuación otra vez un “**ps lxw**”. Observa como ahora hay tres procesos que son la Version2 del intérprete. ¿Por qué ocurre esto? (Observa el código de la función **main** de la Version2 para intentar encontrar la causa).

Esto ocurre porque si ocurre un error en el exec que ejecuta el proceso hijo, no se termina el nuevo proceso, por lo que hay copias creadas por “fork” iguales que el padre. Si se ejecuta un comando válido, el programa sale correctamente.

1. ¿Qué hace la llamada al sistema \_**exit()**? ¿Qué se le pasa cómo parámetro? ¿Qué proceso puede recoger luego dicha información?

\_exit() termina el proceso que lo llama, notifica a los procesos que estén esperando por él, libera recursos, etc. Se le pasa como parámetro un puntero donde se recoga información sobre el proceso terminado, como el código de salida.

1. ¿Qué hace la llamada al sistema **wait()**? (Indica claramente TODO lo que hace).

Suspende la ejecución de un proceso hasta que uno de sus hijos finaliza. Devuelve el PID del proceso hijo.

1. En el directorio **/home/asignaturas/so/Shell** encontrarás la Version3 del intérprete de comandos. Compila y ejecuta la Version3. A continuación, ejecuta varios comandos (**ps**, **ls**, etc) y observa cómo ahora sí sale el *prompt* a continuación de los comandos. Entonces, ¿cómo se ejecutan ahora todos los comandos en la Version3?

Ahora los comandos se ejecutan **en primer plano**, es decir, que hasta que no se termina el proceso hijo, no se hace nada. Esto causa que el prompt se coloque correctamente.

1. Ejecuta la Version3 y repite el ejercicio 4. ¿Por qué ahora no están los procesos que quedaban en la Version2?

Porque ahora se espera a que terminen mediante el comando “wait”.

1. Ejecuta la Version3 y repite el ejercicio 5. ¿Por qué ahora hemos solucionado el problema que se daba con la Version2?

Porque ahora sí que se genera un error y se termina el proceso al ocurrir un fallo en el exec.

1. Sal de nuestro intérprete de comandos. Ejecuta un comando que no exista en el intérprete del sistema (el **bash**) y observa qué pasa. Ejecuta ahora la Version3 y ejecuta también un comando que no exista. ¿Qué diferencia observas entre ambos intérpretes? Soluciona el problema añadiendo código en la función **main**() de la Version3 y entrega dicha Version3 mejorada.

En la versión 3 no se muestra un mensaje de error, simplemente se sale. Para arreglar esto simplemente hay que introducir un cout en la línea 138:

cout << "Error al ejecutar el comando " << comando\_argv[0] << endl;