Laboratorio #3 Procesos

1- trapping to kernel mode es una interrupción donde el sistema pasa a ejecutarse en modo kernel (“root”) para realizar operaciones que solo se pueden realizar en este modo, para después devolverle el control del sistema operativo al usuario.

2- la entrada del manual de llamadas al sistema contiene una descripción de lo que son llamadas al sistema, explicación del funcionamiento y las librerías que usa para el mismo, y una lista con las llamadas al sistema que soporta Linux.

3- Esta entrada contiene una descripción de lo que son llamadas al sistemas, un ejemplo de dos formas de obtener el ID de un proceso de manera diferentes, y una lista de llamadas al sistema junto con la descripción de lo que hacen y el source de esta.

4- No hay punto 4

5- el programa lo que hace es primeramente capturar el id del proceso que realiza la llamada al sistema, y lo imprime en pantalla seguidamente muestra el id del proceso padre que en este caso era el mismo que el que realiza la llamada, porque no se están creando más procesos, y también lo imprime en pantalla. Después mediante la función system ejecuta comandos dentro del programa, el primero es mostrar un calendario con la fecha actual y el segundo es listar archivos por el tamaño de estos.

6- getpid: captura el id del proceso que hace la llamada al sistema.
getppid: captura el id del proceso padre que realiza la llamada al sistema

7- system(“cal”): calendario con la fecha actual.
system(“ls -l .”): listado de archivos organizado por tamaño.

8- la salida del código es el PID del proceso principal, seguidamente se hace un fork a este y muestra el PID del proceso hijo 0, ya que es l valor por defecto que se e asigna a un proceso hijo a menos que a operación falle en ese caso el retorno seria -1, una vez mostrado esto los dos procesos terminan su ejecución.

9- nieto 3, nieto 1, nieto 2, padre.

10- la diferencia radica en el caso donde el proceso padre finaliza su ejecución. en el ejemplo 4 el proceso padre termina simplemente termina su ejecución, y en el ejemplo 6 el proceso padre espera hasta que los demás procesos hijos terminen para poder finalizar su ejecución.

11- Se adjunta el código a la carpeta con nombre: punto11.c.

12- El PID del proceso padre es: 6027.
El PID del proceso hijo es: 6471.
kill -9 6027
Cuando se mata el proceso padre, también mueren todos los procesos hijos del padre que fue asesinado.

13- De los resultados se puede deducir que el proceso que con el que se está trabajando el hijo del proceso principal bash, por lo tanto al matar el proceso padre (“bash”), se están matando no solo el proceso que estábamos trabajando si no todos los procesos que desprenden de este (“hijos”).

14- El anterior programa reemplaza la imagen anterior del proceso, por una imagen de un nuevo proceso. Es decir ejecuta el programa ls que fue pasado como parámetro en el llamado a la función execl.

15- Que no se ejecuta el código en su totalidad, es decir se esperaba que cuando se ejecutara el código se mostrara la ejecución del programa de tres maneras diferentes, pero a cambio de esto solo se ejecutó de una forma y cuando esta se acabó de ejecutar terminó con la ejecución del programa total, sin poder ejecutarse las otras dos formas.

16- Se adjunta el código a la carpeta con nombre: punto16.c.

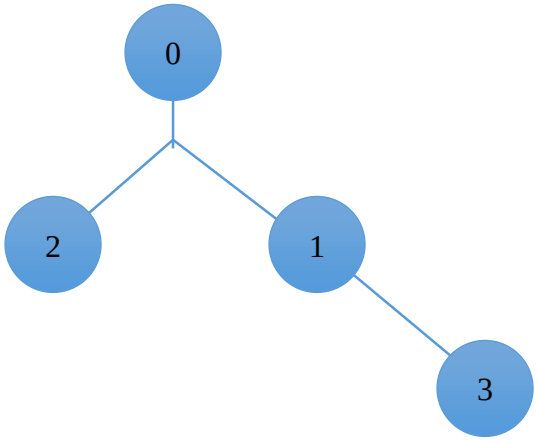
17- Se adjunta el código a la carpeta con nombre: punto171.c, punto172.c, punto173.c.

18- El retorno de la función fork es -1 en caso de que el nuevo proceso no se pueda crear, retorna 0 cuando se trata de un proceso hijo, y retorna un valor de tipo pid_t para expresar PID del nuevo proceso creado.

19- En primera medida, si el fork no se logra hacer de manera correcta se esperaría que saliera un mensaje indicando (“fork fallo”), indicando que no fue posible la creación del nuevo proceso. De lo contrario se esperara una secuencia de salidas con la palabra padre y otra con hijo acompañadas de un número, donde el número del padre ira aumentando de a uno, mientras que el hijo ira aumentando de a tres, estos os van hasta 20.

20- Esto se debe a la veracidad de incremento de la variable num con respecto al proceso, siendo 1 para el hijo y 3 para el padre. Sabemos que las variables definidas antes de hacer el proceso son nuevamente instanciadas y trabajadas particularmente en el proceso nuevo como variables diferentes, es por esto que es posible imprimir diferentes valores de una una variable que aunque en nombre sean semejantes son diferentes.

21-Esperaria que al ejecutar se imprimiera lo siguiente: Hola Mundo !.
La jerarquía que se tomaría en los procesos será:



22- No era la salida que esperaba, pensé que solo se repetiría una vez a palabra Hola Mundo! Pero se repitieron 4 veces. Como los procesos que se crean comparten contextos, y como los dos deben usar el printf, y el proceso que sigue tendrá los datos del proceso anterior, por lo tanto para evitar ese tipo de problemas se debe usar un fflush para poder limpiar el buffer y evitar esos problemas.

23- Se adjunta el código a la carpeta con nombre: punto23.c.

24- sleep (pausa) al proceso hijo durante una cantidad entregada en el parámetro de segundos, en el caso del punto se pausa el proceso durante 5 segundos.

25- El padre del padre según el programa viene siendo el bash que es el intérprete de los comandos.

26- Imprime 1 como el PID del padre porque a momento de la ejecución del proceso hijo el padre ya había terminado su ejecución, por lo tanto el hijo en el instante de la ejecución no tiene un padre asignado (proceso huérfano).

27- El PID que se imprime ahora es e PID del padre del proceso. La función wait sirve para tener en stand by a un proceso con el fin de que no termine su ejecución hasta que sus procesos hijos no acaben. En Status por lo general se retorna el valor del retorno del proceso, para este caso el proceso hijo, por lo tanto retorna 0.

28- Se evidencia que el proceso hijo no ha podido ser liberado, ya que el su proceso padre aun o termina su ejecución, por lo tanto el proceso hijo pasara a tener un estado zombie.

29- La única forma de que el proceso hijo ya terminado sea liberado del PCB, es cuando el proceso padre termina su ejecución de manera normal o realiza el llamado al sistema wait. Por lo tanto el proceso hijo seguirá con el estado de zombie.

30- El primer programa crea un proceso hijo usando fork, a su vez usa la función execl, para ejecutar el comando “ls” que lista los archivos del path donde se indique, en este caso es la dirección de la carpeta raíz. El segundo programa remplaza la ejecución del proceso con la función execlp, donde este ejecuta el comando “ps -ax”.

31- Se adjunta el código a la carpeta con nombre: punto31.c.

32- Se adjunta el código a la carpeta con nombre: punto32.c.

33- Se adjunta el código a la carpeta con nombre: punto33.c.

34- Porque el programa en primera instancia lee el archivo, entonces lo que hace cada uno de esos fork es tener un duplicado de la variable que se creo cuando se abre el buffer de lectura, antes de crear el prime fork lee la primera letra, crea el proceso padre y lee la segunda y el proceso hijo lee la tercera. No el proceso hijo no heredo ningún atributo del proceso padre, solo trabajo con las variables duplicadas que se habían creado antes de que el proceso hijo fuese creado.