Práctica

Programación de Procesos

Ejercicios

1.- Qué interpretación tiene la salida generada por los siguientes proceso ?

```
#include <stdio.h>
#include <unistd.h>
main()  {
        fork();
        printf("Quien soy, el padre o el hijo\n");
        exit(0);
   }
```

```
#include <stdio.h>
#include <unistd.h>
main()  {
        fork();
        fork();
        printf("Soy  el proceso %d\n", getpid());
        exit(0);
   }
```

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main()  {
       pid_t pid;
       if ((pid = fork()) < 0)
             {
                printf("Ha ocurrido un error\n");
               exit(pid);
            }
        printf("Quien soy, el padre o el hijo\n");
        exit(0);
   }
```

2.- Modifique el programa anterior para identificar el hijo y el padre. Lograr que el padre muestre su identificador y el identificador de su padre, y que el hijo muestre su identificador y el identificador de su padre.

3.- Agregar al programa anterior la siguiente línea **system("ps")** que permite la ejecución del comando ("**ps**"). El ps se tiene que ejecutar antes que el **fork.**

4.- Agregar al programa anterior la ejecución de ("**ps**") con el parámetro que permita visualizar el **PPID** de los procesos. El comando lo debe ejecutar el hijo. Imprimir una corrida de ejecución y relacionar los identificadores de procesos.

5.- Convertir el proceso hijo en huérfano; utilizar la llamada al sistema **sleep**. Presentar una corrida del proceso que lo demuestre.

6.- Convertir el proceso hijo en zombie; utilizar la llamada al sistema **sleep**. Presentar una corrida del proceso que lo demuestre.

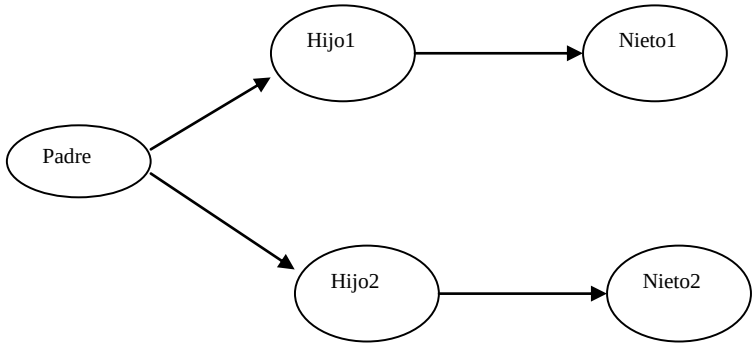
Práctica

7.- Utilizar la llamada al sistema **wait** para que un proceso padre espere la finalización de un proceso hijo.

8.- Cuál es la estructura jerárquica de parentesco de procesos que genera el siguiente programa?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main()  {
        pid_t pid;
        pid = fork();
        if (pid == 0) pid = fork();
        if (pid > 0) pid = fork();
        printf("%d hijo de %d\n",getpid(),getppid());
}
```

9.- Generar utilizando la llamada al sistema **fork**, la siguiente estructura de procesos.



En el proceso PADRE definir un dato local de **main()** antes del **fork** que será heredado por su descendencia, lograr que los HIJOS y los NIETOS modifiquen el dato utilizando alguna operación aritmética, antes de terminar cada proceso debe mostrar por pantalla el valor del dato. Imprimir una corrida de los procesos y analizar los resultados obtenidos.

10.- Realizar un programa utilizando la llamada al sistema **execl,** el programa antes de terminar debe pedir que se presione la tecla ENTER. Probar el programa pasándole como parámetro al **execl** un comando válido y luego pasándole un comando no válido. Cuál es la diferencia en la ejecución?

11.- Realizar un programa utilizando la llamada al sistema fork, el proceso hijo debe ejecutar el comando("ps") utilizando system y en la línea siguiente utilizando execl. Analizar la diferencia entre la salida del comando ("ps") lanzado por el system y la salida del comando ("ps") utilizando la llamada al sistema execl.

12.- Crear una cadena de N procesos, informar por cada proceso los siguientes datos: ID de proceso padre, ID de proceso actual, ID de proceso hijo. Generar un archivo con la salida y verificar la existencia de procesos zombies y huérfanos. N es un valor que se ingresa desde la línea de comandos e indica la cantidad de procesos de la cadena.

13.- Crear un abanico de N procesos, informar por cada proceso los siguientes datos: ID de proceso padre, ID de proceso actual, ID de proceso hijo. Generar un archivo con la salida y verificar la existencia de procesos zombies y huérfanos. N es un valor que se ingresa desde la línea de comandos e indica la cantidad de procesos del abanico.

14.- Verificar mediante la creación de un proceso padre y su correspondiente hijo que en el instante de la creación del hijo, este hereda el contenido de las variables del padre, luego cualquier modificación de las

mismas ya no son vistas por el otro proceso. Explique porqué? Cuál es la única variable que no se hereda su contenido?

15.- Realizar un programa que le permita verificar, que cuando un proceso hijo termina antes que el padre, y el padre está esperando su terminación con la llamada al sistema wait, si el hijo queda en estado zombie o no.

16.- Crear una cadena de N procesos, y sincronizar la terminación de los mismos en el orden inverso al que fueron creados. Informar por cada proceso los siguientes datos: ID de proceso padre, ID de proceso actual, ID de proceso hijo. Generar un archivo con la salida y verificar la existencia de procesos zombies y huérfanos. N es un valor que se ingresa desde la línea de comandos e indica la cantidad de procesos de la cadena.

17.- Realizar un programa llamado mishell que permita ejecutar comandos. Cuando se ejecuta mishell aparece el prompt msh$ luego escribir el comando a ejecutar, cuando se termina de ejecutar vuelve a aparecer el prompt msh$. (NO USAR system( )).