Ejercicio 3

Crea un programa en C que:

- Cree un Proceso Hijo y un Proceso Nieto (Hijo del Hijo)
- Cada Proceso debe de escribir un texto que diga “Soy el …”

Solución:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(){

    int pid = 0;
    int pid2 = 0;

    pid = fork();

    if (pid == 0){

        pid2 = fork();

        if (pid2 == 0){
            printf("Soy el nieto\n");
        } else {
            printf("Soy el hijo\n");
        }

    } else {
        printf("Soy el padre\n");
    }

    exit (0);
}
```

Ejercicio 4

Crea un programa en C que:

- Cree un Proceso Padre y un Proceso Hijo
- Defina una variable entera con valor de al principio del programa.
- El Padre incrementará el valor en 5. El Hijo lo decrementará en 5.
- Muestra los valores por pantalla.
- ¿Por qué no se confunde el programa con las sumas y restas?

Solución:

```
int main(){
    int hijo;
    int num = 6;
    hijo = fork();
    if (hijo != 0) {
        // HIJO
        num = num - 5;
        printf("HIJO: num=%d \n", num);
    }else{
        // PADRE
        num = num + 5;
        printf("PADRE: num=%d \n", num);
        wait(NULL);
    }
    exit(0);
}
```

No se confunde porque al "clonarse" el proceso para que haya un Hijo, también 'clona' la variable y, por tanto, cada proceso tiene la suya.

Ejercicio 5

Copia el siguiente programa y ejecútalo. ¿Qué se muestra por pantalla?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(){
      int estado, hijo;
      hijo = fork();
      if (hijo != 0) {
            wait(&estado);
            printf("PADRE: pid=%d ppid=%d user-id=%d \n", getpid(),
            getppid(), getuid());
            printf("Codigo de retorno del hijo: %d\n", estado);
      }else{
            printf("HIJO: pid=%d \n", getpid());
      }
      exit(0);

}
```

Solución:

HIJO pid: 3363

PADRE pid: 3362, ppid: 3360, user-id:1000

Código de retorno del hijo: 0

Responde:

- ¿Que se ejecuta antes? ¿El padre o el hijo? ¿Por qué?

  Solución: El hijo, porque el padre se queda esperando a que el hijo termine

- En la instrucción wait(&estado), ¿qué es lo que se recibe en la variable estado?

  Solución: El código de finalización del hijo, que en este caso es cero (por el exit (0))

Ejercicio 6

Realiza un programa en C que cree una carpeta llamada “trabajo”

Solución:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
main()
{
    // Llamada a una function exec
    execl("/bin/mkdir", "mkdir", "/home/Desktop/work", NULL);
    printf("No deberia llegar aqui\n");
    exit(-1);
}
```

Ejercicio 7

Copia el siguiente programa, ejecútalo y apunta el PID.

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
void trapper(int);
int main(){
      int i;
      for(i=1;i<=64;i++)
            signal(i, trapper);
            printf("Identificador del proceso: %d\n", getpid() );
            pause();
            printf("Continuando...\n");
            printf("Termino.\n");
            return 0;
      }
void trapper(int sig){
      printf("\nSeñal que he recogido: %d\n", sig);
}
```

Responde:

- Este Proceso no termina nunca, porque está en pausa. Desde otra terminal, envíale la señal SIGUSR1. ¿Cuál es el comando que has utilizado?

  Solución:

  kill -USR1 4545

  Señal que he recogido 10

- ¿Qué es lo que pasa si un proceso recibe una Señal que no trata?

  Solución: Por defecto, "matará" el proceso

- Cambia el programa para que sólo recoja las señales del 1 al 9.

  Solución: Por ejemplo → for(i=1; i<=9; i++)

- Ejecútalo y mándale la señal SIGUSR1. ¿Cuál es el comando que has utilizado?

  Solución: kill -USR1 4549

- ¿Qué ha pasado al enviarle la señal? ¿Por qué?

  Solución: El programa termina porque no está preparado para recoger la señal SIGUSR1 (10), por lo que ejecutará el comportamiento por defecto.

Ejercicio 8

Crea un programa en C que:

- Cree un proceso Hijo y un proceso Nieto (hijo del hijo)
- Tenga dos PIPE
- El padre envíe un mensaje al Hijo; y el Hijo se lo pasará al Nieto.

Solución:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>

#define SIZE 256

// Comunicación Padre - Hijo - Nieto

int main (){

    int pid, pid2;

    int pipePadreHijo [2];
    int pipeHijoNieto [2];

    char buffer [SIZE];
    char mensaje [] = "Lorem ipsum dolor sit amet \0";

    pipe (pipePadreHijo);
    pipe (pipeHijoNieto);

    pid = fork (); // Creamos al hijo

    if (pid == 0){
        // Hijo

        pid2 = fork (); // Creamos al nieto

        if (pid2 == 0){
            // Nieto
```

```
34          if (pid2 == 0){
35              // Nieto
36
37              // Dado que el hijo envía al nieto...
38              close (pipeHijoNieto[1]);
39              printf ("Nieto recibiendo del Hijo: ");
40              read (pipeHijoNieto[0], buffer, strlen (buffer));
41              printf ("%s, de tamaño %d \n", buffer, strlen (buffer));
42
43              printf ("Nieto finaliza... \n");
44          } else {
45              // Hijo
46
47              // Dado que el padre envía al hijo...
48              close (pipePadreHijo[1]);
49              printf ("Hijo recibiendo del Padre: ");
50              read (pipePadreHijo[0], buffer, strlen (buffer));
51              printf ("%s, de tamaño %d \n", buffer, strlen (buffer));
52
53              // Dado que el hijo envía al nieto...
54              printf ("Hijo enviando a Nieto... \n");
55              close (pipeHijoNieto[0]);
56              write (pipeHijoNieto[1], buffer, strlen (buffer));
57
58              // Esperamos a que el finalice el Nieto
59              wait (NULL);
60
61              printf ("Hijo finaliza... \n");
62          }
63      } else {
64          // Padre
65
66          // Dado que el padre envía al hijo...
67          printf ("Padre enviando a Hijo... \n");
68          close (pipePadreHijo[0]);
69          write (pipePadreHijo[1], mensaje, strlen (mensaje));
70
71          // Esperamos a que el finalice el Hijo
72          wait (NULL);
73
74          printf ("Padre finaliza... \n");
75      }
76      exit(0);
77  }
78
```