



INSTITUTO POLITÉCNICO
NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
ESCOM



SISTEMAS OPERATIVOS

PROFESOR: RANGEL GONZÁLEZ JOSUÉ

GRUPO: 4CV2

INTEGRANTES:

- GONZÁLEZ OLOARTE DIEGO ENRIQUE
- GUTIÉRREZ JIMÉNEZ CINTHIA NAYELLI
- LÓPEZ BARRERA DANIEL

Introducción

Los procesos de un sistema Linux (y en general en los sistemas basados en UNIX), tienen una estructura jerárquica en donde un proceso (padre) puede crear un nuevo proceso (hijo), y así sucesivamente (un proceso padre puede tener muchos hijos) lo cual forma una estructura de tipo árbol, en donde la raíz es el nodo padre principal (en un ambiente Linux, el nodo raíz será el shell desde el cual se ejecuta el proceso).

La ejecución de la función fork crea procesos hijos que son una copia fiel del padre después del punto de ejecución de fork; y a partir de ese punto, aun cuando ambos procesos poseen el mismo código, posiblemente no contendrán los mismos datos.

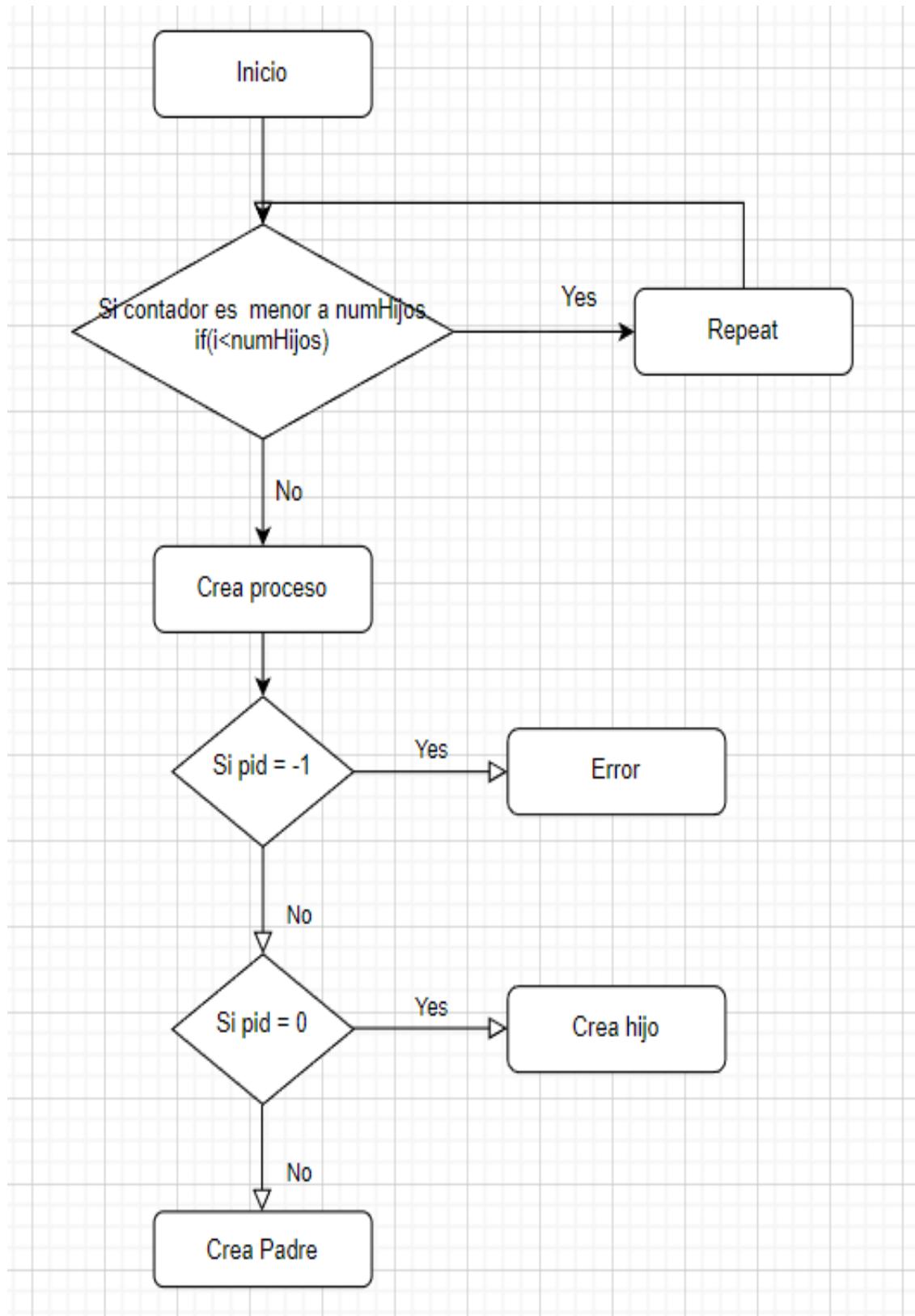
Tanto padre como hijo son exactamente iguales, tienen el mismo contenido en las variables pero no las comparten: es decir, una vez creados, siguen su ejecución de manera independiente, y lo que los hace diferentes es su identificador. Esto último es muy importante, ya que significa que aún cuando hay una relación de parentesco entre los procesos, ambos procesos son aparentemente independientes.

Problemática

Partiendo del ejercicio realizado con anterioridad en clase, en el cual a partir de la sentencia fork se generaba un árbol controlado en donde el primer padre generaba dos hijos y a su vez los hijos generaban otro hijo cada uno de ellos, en la práctica lo que se busca es que a partir de esa estructura el padre genere 2 y 3 hijos respectivamente de manera dinámica, con un número de niveles ingresado por entrada estándar.

Desarrollo

Diagrama de Flujo



Elementos más importantes

La parte más importante es cuando se ejecuta el procesoHijo(niveles), ya que la variable de niveles la pasamos como argumento y comenzará a crear los procesos hijos dependiendo el nivel.

```
● ● ●
1 void procesoHijo(char *niveles){
2     pid_t miID = getpid();
3     pid_t padreID = getppid();
4
5     if(miID == padreID+1){
6         //Hijo que creará dos hijos en cada nivel
7         printf("\nYo soy el proceso hijo de la izquierda, mi ID es %d El ID del padre es %d\n", miID, padreID);
8
9         char *argumentos [] = {"./clonacion_proceso", niveles, "2"};
10        execvp(argumentos[0], argumentos);
11        fprintf(stderr, "Failed to execvp() '%s' (%d: %s)\n", argumentos[0], errno, strerror(errno));
12    }
13    else{
14        //Hijo que creará 3 hijos en cada nivel
15        printf("\nYo soy el proceso hijo de la derecha, mi ID es %d El ID del padre es %d\n", miID, padreID);
16
17        char *argumentos [] = {"./clonacion_proceso", niveles, "3"};
18        execvp(argumentos[0], argumentos);
19        fprintf(stderr, "Failed to execvp() '%s' (%d: %s)\n", argumentos[0], errno, strerror(errno));
20    }
21 }
```

Posteriormente, ejecutará la llamada al sistema “execv” la cual ejecuta el programa “clonacion_proceso” en el cual está la otra parte importante, puesto que en ésta sección de código creará los procesos hijos respectivamente.

```
● ● ●
1 for (i = 1; i <= niveles; i++){
2     int numeroHijos;
3     for(numeroHijos=0; numeroHijos<hijos; numeroHijos++){
4         pid=fork();
5         if (pid <= 0) break;
6     }
7     if (pid > 0) break;
8 }
```

Evidencia

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6 #include <errno.h>
7 #include <string.h>
8
9 void errorEnFork();
10 void procesoHijo(char *niveles);
11 void procesoPadre(pid_t pid);
12
13 int main (int argc, char *argv[])
14 {
15     pid_t pid;
16     int status;
17
18     int i;
19     char *niveles;
20
21     niveles = argv[1];
22
23     if (argc != 2) {
24         fprintf(stderr, "Insuficientes parámetros");
25         return 1;
26     }
27
28     int numeroHijos = 2;
29
30     //Creamos 2 hijos
31     for (size_t i = 0; i < numeroHijos; i++){
32
33         pid = fork();
34
35         if (pid == -1){
36             errorEnFork();
37         }
38         else if (pid == 0){//Ejecución del hijo
39
40             //Cada hijo, crea 1 hijo
41             procesoHijo(niveles);
42
43             break;
44         }
45         else{
46             procesoPadre(pid);
47             wait(NULL);
48         }
49
50     for (size_t i = 0; i < numeroHijos; i++){
51         wait(NULL);
52     }
53
54     exit(0);
55 }
56
57 void errorEnFork(){
58     printf("\nNo se pudo crear un proceso hijo");
59 }
60
61 void procesoPadre(pid_t pid){
62     pid_t miID = getpid();
63     pid_t hijoID = pid;
64     printf("\nYo soy el proceso padre, mi ID es: %d El ID del hijo es %d\n", miID, hijoID);
65 }
66
67 void procesoHijo(char *niveles){
68     pid_t miID = getpid();
69     pid_t padreID = getppid();
70
71     if(miID == padreID+1){
72         //Hijo que creará dos hijos en cada nivel
73         printf("\nYo soy el proceso hijo de la izquierda, mi ID es %d El ID del padre es %d\n", miID, padreID);
74
75         char *argumentos [] = {"./clonacion_proceso", niveles, "2"};
76         execvp(argumentos[0], argumentos);
77         fprintf(stderr, "Failed to execvp() '%s' (%d: %s)\n", argumentos[0], errno, strerror(errno));
78     }
79     else{
80         //Hijo que creará 3 hijos en cada nivel
81         printf("\nYo soy el proceso hijo de la derecha, mi ID es %d El ID del padre es %d\n", miID, padreID);
82
83         char *argumentos [] = {"./clonacion_proceso", niveles, "3"};
84         execvp(argumentos[0], argumentos);
85         fprintf(stderr, "Failed to execvp() '%s' (%d: %s)\n", argumentos[0], errno, strerror(errno));
86     }
87 }
```

clonacion_proceso.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5 #include <math.h>
6
7 void infoProceso(int i, pid_t pid);
8
9 int main(int argc, char *argv[]){
10
11     pid_t pid;
12     int i, niveles, hijos;
13
14     niveles = atoi(argv[1]);
15     hijos = atoi(argv[2]);
16     pid=-1;
17
18     for (i = 1; i <= niveles; i++){
19         int numeroHijos;
20         for(numeroHijos=0; numeroHijos<hijos; numeroHijos++){
21             pid=fork();
22             if (pid <= 0) break;
23         }
24         if (pid > 0) break;
25     }
26
27     while(wait(NULL) > 0) ;
28
29     infoProceso(i, pid);
30
31     return 0;
32 }
33
34 void infoProceso(int i, pid_t pid){
35     printf("Nivel:%d ID de proceso:%ld ID del padre:%ld ID del hijo:%ld\n",i, (long)getpid(), (long)getppid(), (long)pid);
36 }
```

Conclusiones

González Oloarte Diego Enrique: La práctica significó un problema a la hora de realizarla, puesto que los integrantes del equipo no contábamos con los conocimientos necesarios para realizarla, sin embargo con el uso de algunas herramientas de consulta nos fue posible ejecutarla. No obstante, existe un pequeño error en nuestro programa, el cual no fuimos capaces de solucionar.

López Barrera Daniel: Considero, que la elaboración de la práctica resultó un poco problemática, debido a que escaseamos en tiempo y conocimientos que no se profundizaron en clase. Sin embargo, logramos terminar una parte significativa.

Gutiérrez Jiménez Cinthia Nayelli: El desarrollo de la práctica resultó bastante problemática ya que de manera personal me faltó revisar y entender más respecto al tema, por lo que al intentar implementar los conocimientos, no resultó de manera correcta.

