Procesos mediante fork

José Luis Quiroz Fabián

1 Introducción

Los procesos de un sistema Linux (y en general en un sistema basado en Unix) tienen una estructura jerárquica, de manera que un proceso puede crear un nuevo proceso y así sucesivamente. Cuando un proceso crea un nuevo proceso, al proceso creador se llama *proceso padre* y al proceso creado se le llama *proceso hijo*. Para crear procesos, el sistema operativo Linux proporciona la llamada al sistema *fork()*.

Cabecera:

```
1
2     #include <unistd.h>
3     #include <sys/types.h>
4
5      int fork(void);
```

La llamada *fork()* crea (clona) un nuevo proceso exactamente igual al proceso que invoca la función. En caso de error, la función devuelve el valor -1 y no se crea el proceso hijo. Si no hay problema en la creación, el proceso padre (el proceso que realizó la llamada) obtiene el identificador (*pid*) del proceso hijo que acaba de nacer, y el proceso hijo recibe el valor 0.

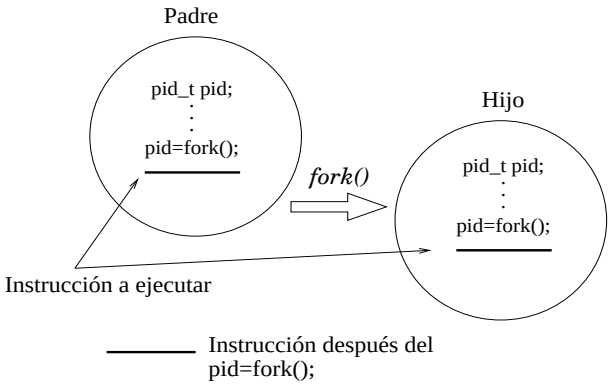


Figure 1: Esquema de creación de procesos con fork

2 Comunicación mediante wait y exit

Debido al parentesco que adquieren un par de procesos cuando uno crea al otro estos se pueden comunicar *sólo una vez* haciendo uso de las funciones *exit* y *wait*.

Cabecera:

```
#include <stdlib.h>

void exit(int status);
```

Comportamiento de la función

Cuando un proceso ejecuta *exit()* termina su ejecución. El valor de *status* se le regresa al proceso padre del proceso que ejecuta *exit* para que este pueda conocer como terminó su proceso hijo.

Cabecera:

```

#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int status);
```

Comportamiento de la función

Cuando un proceso ejecuta *wait()* obtiene el estado con el que termina uno de sus procesos hijo. Cuando un proceso ejecuta *wait* y ninguno de sus procesos hijos ha terminado, éste se queda bloqueado. El valor que regresa esta función es el *pid* del proceso hijo que terminó.

La función *wait* almacena la información del estado con el que terminó un proceso en *parte* de la memoria de la variable *status*. Esta información puede ser evaluada/obtenida usando las macros:

- **WIFEXITED(status)** regresa un valor distinto de cero si el hijo terminó de forma normal.
- **WEXITSTATUS(status)** evalúa o regresa el contenido de los ocho bits menos significativos, los cuales regresan el valor con el que termina el proceso hijo.

3 Ejemplos usando fork

3.1 Creación de un padre con su hijo

Este primer ejemplo se muestra como crear un proceso con un sólo hijo.

```
#include <sys/types.h>
#include <stdio.h>
```

```c
#include <unistd.h>
#include <stdlib.h>
 main(){
  int valor=0;
  pid_t pid;
  pid = fork();
  switch(pid){

     case 0:
          valor = getpid();
          break;
     default:
          printf("Valor de la variable: %d PID de mi hijo: %d\n",
    valor,pid);
          break;
  }
  sleep(30);
  exit(0);
}
```

Para visualizar el árbol puede utilizar el comando *pstree*. El comando permite *pstree* visualizar todos los árboles de procesos del sistema. Si se usa la opción *p* del comando, se visualiza el árbol de procesos del proceso cuyo identificador es *p*.

Para ejecutar el siguiente programa abra dos terminales. En una terminal compila y ejecuta el programa:

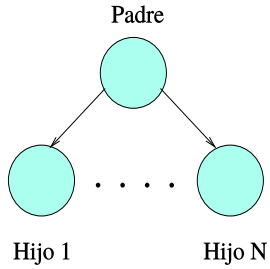
```
[user@pc]$ gcc ejemplo1.c -o ejemplo1
[user@pc]$ ./ejemplo1
```

En la otra terminal visualiza el árbol mediante la secuencia de comandos:

```
[user@pc]$ ps -eo pid,comm  | grep "ejemplo1"| head -n 1 | awk '{print $1}'|
           xargs pstree -c -p
```

3.2 Un proceso padre con N hijos

En este ejemplo se genera la siguiente topología de procesos:



Como se muestra en la figura anterior, se tiene un proceso padre con N hijos. La solución de este problema se presenta a continuación con N = 4.

```c

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define N 4
/*
Entrada:
Salida:
Descripción:Creación de N hijos de un padre
*/

main()
{
        int i;
        pid_t pid;


        for(i=0;i<N;i++){

                switch(pid=fork()){
                        case 0:
                                printf("Soy el proceso hijo: %d y mi padre es %d \n",getpid(),getppid());
                                break;
                        case -1:
                                printf("Error en la creación del proceso \n");
                                exit(0);
                        default:
                                printf("Soy el proceso padre: %d \n",getpid());
                }
                if(pid==0)
                    break;
        }
        sleep(10);
}
```

Ejercicio: Visualice el árbol de procesos en una terminal.

3.3 Árbol binario de procesos

```c

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

/*
 * Entrada n Profundidad del arbol:
 * Salida:
 * Descripción:Creación de una jerarquia binaria
 * */

```

```
main(int argc, char **argv)
{
        int i,raiz,izq,der,n;
        pid_t pid;

        if(argc<2){
           printf("Error: indicar la profundidad del arbol\n");
           exit(0);
         }
        n=atoi(argv[1]);
        raiz=getpid();
        for(i=0;i<n;i++){

              switch(fork()){
                   case 0:
                        break;
                   case -1:
                        printf("Error en la creación del proceso \n")
    ;
                        exit(0);
                   default:
                        switch(fork()){
                             case 0:
                                  break;
                             case -1:
                                  printf("Error en la creación del
    proceso \n");
                                  exit(0);
                             default:
                                  wait(&izq);
                                  wait(&der);
                                  if(raiz==getpid()){
                                       printf("#Procesos creados: %d\n
    ",WEXITSTATUS(izq)+
                                       WEXITSTATUS(der)+1);
                                       exit(0);
                                  }else
                                       exit(WEXITSTATUS(izq)+
    WEXITSTATUS(der)+1);

                        }
              }
        }
        sleep(30);
        exit(1);
}
```

Ejercicio: Visualizar el árbol binario para diferentes valores de n.

4 Ejercicios

1. Realizar un árbol impar de procesos organizados linealmente. En base a un número N se deberá crear un árbol de procesos que presentará un patrón semejante al de la Figura 2. En esta figura se supone que N=7. Al final el proceso *raíz* deberá mostrar el número total de procesos creados.

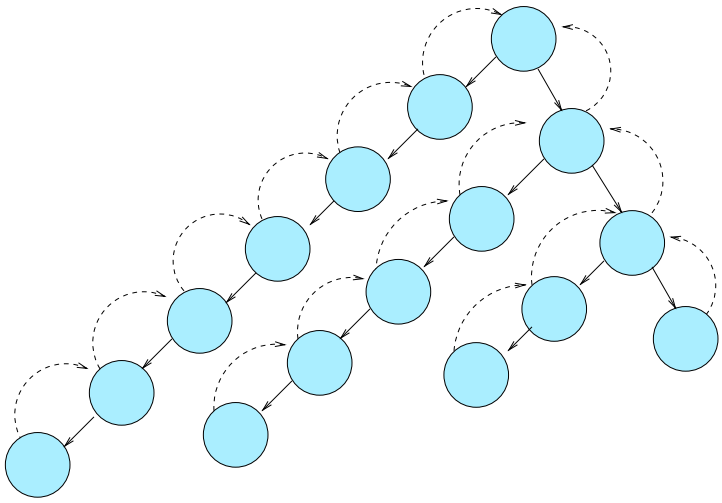


Figure 2: Árbol de procesos impares

2. Se requiere conocer la información de un conjunto de archivos y carpetas en un directorio particular. Al ejecutar un programa llamado **analizar** se le pasa como parámetro una ruta de un directorio. El programa debe mostrar la meta-información de cada archivo (tamaño, último acceso, última modificación, permisos, etc.). Si hay un directorio o más por cada directorio se crea un proceso que sigue el mismo procedimiento (muestra la meta-información de cada archivo) y se crean tantos procesos como se necesiten. Además, después de hacer el análisis completo el último proceso en morir es la raíz.