Práctica 2: Fork y Funciones EXEC

Adriana Pérez Espinosa
José Luis Quiroz Fabián

Febrero 2017

1 Creación de procesos: fork()

Los procesos de un sistema Linux (y en general en un sistema basado en Unix) tienen una estructura jerárquica, de manera que un proceso (proceso padre) puede crear un nuevo proceso (proceso hijo) y así sucesivamente. Para el desarrollo de aplicaciones con varios procesos, el sistema operativo Linux proporciona la función *fork()*.

Cabecera:

```
#include <unistd.h>

int fork(void);
```

fork() crea un nuevo proceso exactamente igual (mismo código) al proceso que invoca la función. Ambos procesos continúan su ejecución tras la llamada *fork()*. En caso de error, la función devuelve el valor -1 y no se crea el proceso hijo. Si no hubo ningún error, el proceso padre (que realizó la llamada) obtiene el *pid* (identificador) del proceso hijo que acaba de nacer, y el proceso hijo recibe el valor 0. Tanto el proceso padre como el proceso hijo continúan su ejecución en la línea siguiente después del llamado *fork*.

2 Ejercicios fork

a) En este ejercicio por medio de parámetros del *main* se ingresará la profundidad p y anchura n (de los nodos) del árbol. En base a esta profundidad y anchura se deberá crear un árbol del procesos (un árbol n-binario de procesos) como el que se muestra en la Figura 1. Cada proceso (excepto los procesos hojas) debe regresar el número de sus descendientes + 1 a su proceso padre, al final, el primer proceso imprimirá el número de procesos en el árbol.

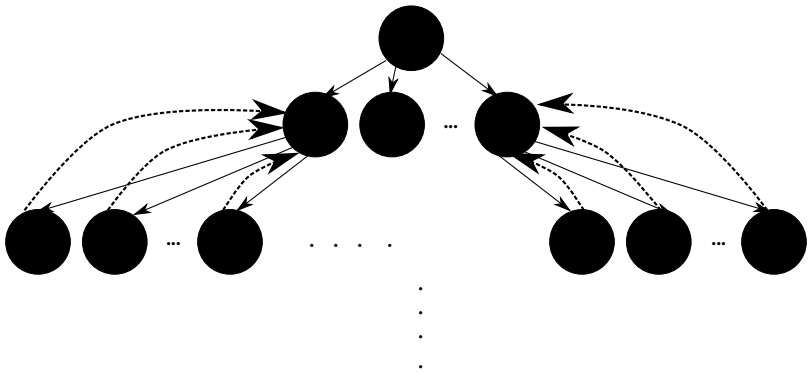

Figure 1: Árbol de procesos n-nario

b) En este ejercicio el primer proceso (el proceso raíz) le pedirá al usuario un número impar ≥ 1. En base a este número se deberá crear un árbol de procesos el cual presentará un patrón semejante al de la Figura 2. En esta figura se supone que k=7. Al final el primer proceso deberá imprimir el número total de procesos creados.

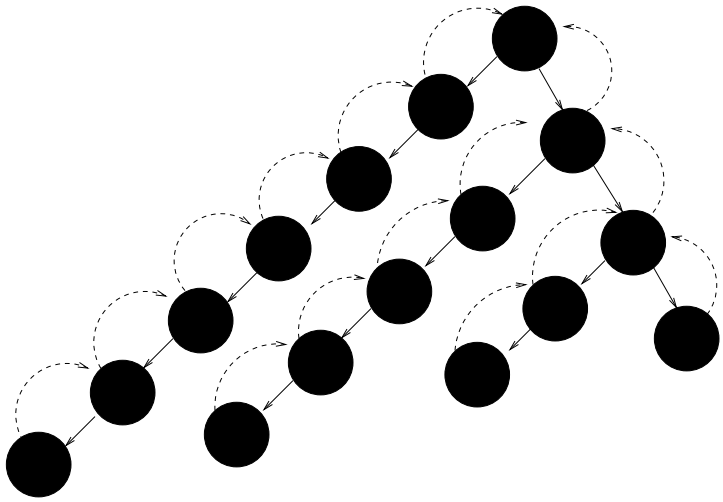

Figure 2: Árbol de procesos impares

3 Funciones Exec

Existe toda una familia de funciones *exec* que pueden ser utilizadas para ejecutar programas. Dentro de esta familia, cada función tiene una interfaz propia, pero todas tiene aspectos comunes y obedecen al mismo tipo de funcionamiento.

Básicamente, el resultado que se consigue con estas funciones es cargar un programa en la zona de memoria del proceso que ejecuta la llamada, sobreescribiendo los segmentos del programa antiguo con los del nuevo. El contenido del contexto del nivel de usuario del proceso que llama a *exec* deja de ser accesible y es reemplazado de acuerdo con el nuevo programa. Es decir, el programa viejo es sustituido por el nuevo y nunca retornaremos a él para proseguir su ejecución, ya que es el nuevo programa el que pasa a ejecutarse.

4 Ejemplos

4.1 Ejemplo1: Gnome Terminal

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

/*
Entrada: -----
Salida: ------
Descripción:Ejecuta la llamada al sistema execv invocando al programa
        gnome-terminal con un conjunto de parámetros
*/
void ejecutar_execv(){

  char *para[6];

  para[0]="gnome-terminal";
  para[1]="-x";
  para[2]="bash";
  para[3]="-c";
  para[4]="ls; read -p 'Presiona la tecla [Enter] para continuar ...'";
  para[5]=NULL;
      if(execv("/usr/bin/gnome-terminal",para)==-1)
          printf("Error al ejecutar execv\n");


}
/*
Entrada: -----
Salida: ------
Descripción:Ejecuta la función ejecutar_execl() o ejecutar_execv() dependiendo
        la opción del usuario.
*/
main(){

    ejecutar_execv();

    printf("Se ha terminado la ejecucion.....\n");
}
```

Figure 3: Ejemplo 1: Gnome Terminal

4.2 Ejemplo2: Xterm

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


/*
Entrada: -----
Salida:  La opcion seleccionada por el usuario
Descripcion:  Menu para seleccionar la funcion a ejecutar
*/
int menu(){
int opcion;
        printf("1------ejecutar execl\n");
        printf("2------ejecutar execv\n");
        scanf("%d",&opcion);

return opcion;
}


/*
Entrada: -----
Salida: ------
Descripcin:Ejecuta la llamada al sistema execl invocando al programa
           xterm con un conjunto de parametros
*/
void ejecutar_execl(){
   if(execl("/usr/bin/xterm", "xterm","-fg","green","-bg","black","-cr","yellow",NULL)==-1)
        printf("Error al ejecutar execl\n");
}


/*
Entrada: -----
Salida: ------
Descripcion:Ejecuta la llamada al sistema execv invocando al programa
           xterm con un conjunto de parametros
*/
void ejecutar_execv(){
  char *para[6];

  para[0]="xterm";
  para[1]="-fg";
  para[2]="green";
  para[3]="-fn";
  para[4]="9x15";
  para[5]=NULL;
       if(execv("/usr/bin/xterm",para)==-1)
           printf("Error al ejecutar execv\n");

}
/*
Entrada: -----
Salida: ------
Descripcion:Ejecuta la funcion ejecutar_execl() o ejecutar_execv() dependiendo
           la opcion del usuario.
*/
main(){
        switch(menu())
        {
                case 1:
                    ejecutar_execl();
                    break;
                case 2:
                    ejecutar_execv();
                    break;
        }

}
```

Figure 4: Ejemplo 2: Xterm

Modifique el programa anterior para que un proceso hijo (mediante fork) ceda sus recursos para ejecutar las terminales xterm. El proceso padre debe seguir permitiendo lanzar varias terminales.