Práctica 1: Concurrencia y fork

José Luis Quiroz Fabián

1 Objetivos

Los objetivos buscados en esta práctica son los siguientes:

- Aprender el concepto de concurrencia.
- Aplicar el concepto de concurrencia haciendo uso de la función *fork*.

2 Concurrencia

Un programa ordinario consiste en declaraciones de datos e instrucciones en un lenguaje de programación. Las instrucciones son ejecutadas de manera *secuencial* en una computadora. Un *programa concurrente* es un conjunto de programas secuenciales los cuales son ejecutados en paralelismo real o bien abstracto (por ejemplo en un sistema multiproceso con un sólo procesador). En esta práctica estudiaremos el concepto de concurrencia haciendo uso de la función *fork*. Primero mostramos como trabaja dicha función. Posteriormente mostramos algunos ejemplos usando *fork*. Finalmente terminamos con 2 ejercicios para el (los) alumno(s).

3 Creación de procesos: fork()

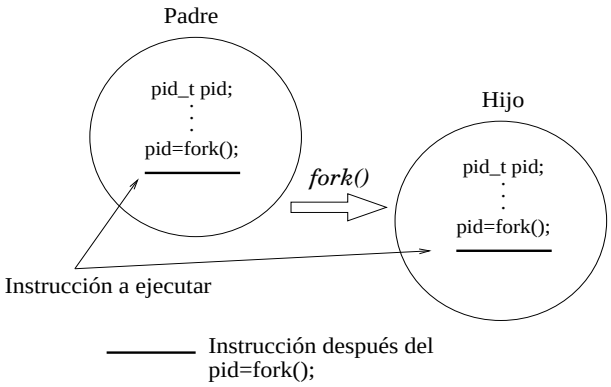
Los procesos de un sistema Linux (y en general en un sistema basado en Unix) tienen una estructura jerárquica, de manera que un proceso (proceso padre) puede crear un nuevo proceso (proceso hijo) y así sucesivamente. Para el desarrollo de aplicaciones con varios procesos, el sistema operativo Linux proporciona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

Cabecera:

```
#include <unistd.h>

int fork(void);
```

Descripción del comportamiento función

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* como se ilustra en la siguiente figura:



4 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 *wait* y *exit*.

Cabecera:

```
#include <stdlib.h>

void exit(int status);
```

Descripción del comportamiento función

exit() termina al proceso que la manda a invocar. El valor de *status* se le regresa al proceso padre para que este pueda conocer como terminó su proceso hijo.

Cabecera:

```
#include <stdlib.h>

pid_t wait(int status);
```

Descripción del comportamiento función

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.

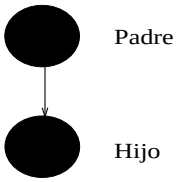
La función *wait* almacena la información de estado con el que terminó un proceso hijo en la memoria apuntada por *status*. Esta información puede ser evaluada usando las macros:

- **WIFEXITED(status)** es distinto de cero si el hijo terminó normalmente.
- **WEXITSTATUS(status)** evalúa los ocho bits menos significativos del código de retorno del hijo que terminó,que podrían estar activados como el argumento de una llamada a exit() o como el argumento de un return en el programa principal.

5 Ejemplos usando fork

5.1 Creación de un padre con su hijo

Este primer ejemplo se muestra como crear un proceso con un sólo hijo. Gráficamente la solución la podemos representar como sigue:



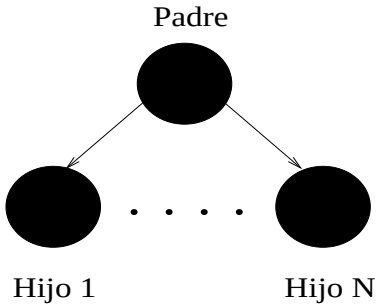
La solución en código sería la siguiente:

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

/*
Entrada:
Salida:
Descripción:Creación de un proceso padre con un hijo
*/

main()
{
       int i;

       switch(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());

    }
    sleep(10);
}
```

5.2 Ejemplo: estructura lineal de procesos (descendientes de procesos)

En este ejemplo se busca generar la siguiente topología de procesos:



Como se muestra en la figura anterior, se tiene un proceso padre con N hijos. Las lineas punteadas representan comunicación entre los proceso (un proceso hijo le envía a su padre en que estado terminó). La solución de este problema se presenta a continuación:

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

#define N 4
/*
Entrada:
Salida:
Descripción:Creacion de procesos
*/

main()
{
    int i,status=0;
    pid_t pid,pid_raiz;

    pid_raiz=getpid();
```

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

             if((pid=fork())==0){

                    printf("Soy el proceso hijo: %d y mi padre es %d \n",getpid(),getppid());
                    break;

             }else{
                   if(pid==-1){

                       printf("Error en la creación del proceso \n");
                       exit(1);

                   }else{
                       printf("Soy el proceso padre: %d \n",getpid());

                   }
             }
        }
        sleep(3);
        exit(1);

}
```

5.3 Compilación y ejecución

Los programas ejemplos de esta práctica se encuentran en la página del curso. Después de descargarlos para compilarlos y ejecutarlos realice lo siguiente: Compilación: Desde un interprete de comandos ejecute

gcc fork.c -o fork

Ejecución: Desde un interprete de comandos ejecute

./fork

6 Ejercicios

a) Generar un árbol lineal de procesos de profundidad N+1.

b) Utilice wait, exit y WEXITSTATUS para contar los procesos del ejercicio a.

c) En este ejercicio el proceso raíz le pedirá la profundidad y amplitud (n) al usuario. En base a esta profundidad 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.

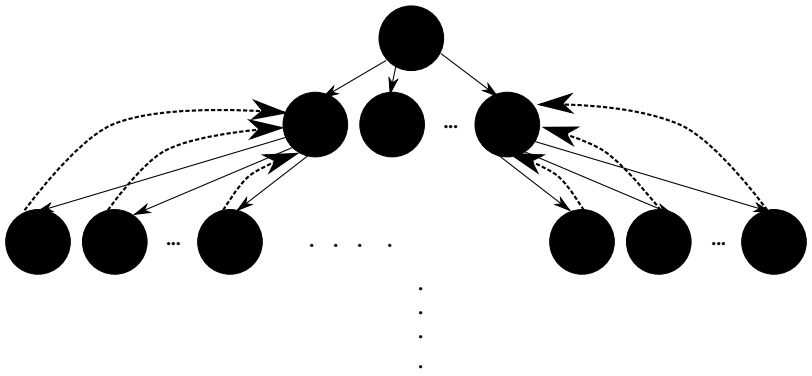


Figure 1: Árbol de procesos n-nario

d) En este ejercicio el primer proceso (el proceso raíz) le pedirá al usuario un número de la forma. 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 numero=6. Al final el primer proceso deberá imprimir el número total de procesos creados.

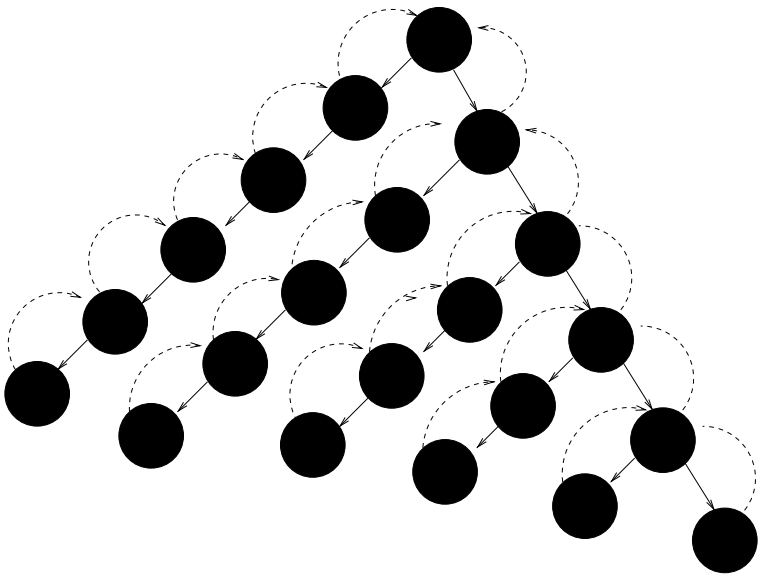


Figure 2: Árbol de procesos lineales