Práctica 1

Oscar Daniel Juárez Cruz

March 20, 2018

1 Introducción

En un sistema multiprogramado o de tiempo compartido, un proceso es la imagen en memoria de un programa, junto con la información relacionada con el estado de su ejecución. Un programa es una entidad pasiva, una lista de instrucciones; un proceso es una entidad activa, que –empleando al programa– define la actuación que tendrá el sistema.

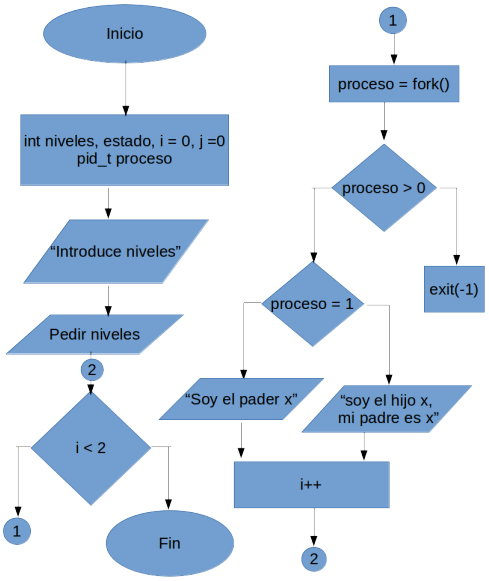
Un proceso, a lo largo de su vida, alterna entre diferentes estados de ejecución. Éstos son:

- Nuevo: Se solicitó al sistema operativo la creación de un proceso, y sus recursos y estructuras están siendo creadas.
- Listo: Está listo para iniciar o continuar su ejecución pero el sistema no le ha asignado un procesador.
- En ejecución: El proceso está siendo ejecutado en este momento. Sus instrucciones están siendo procesadas en algún procesador.
- Bloqueado: En espera de algún evento para poder continuar su ejecución (aun si hubiera un procesador disponible, no podría avanzar).
- Zombie: El proceso ha finalizado su ejecución, pero el sistema operativo debe realizar ciertas operaciones de limpieza para poder eliminarlo de la lista.
- Finalizado: El proceso terminó de ejecutarse; sus estructuras están a la espera de ser limpiadas por el sistema operativo.

2 Desarrollo

La práctica consiste en crear un árbol de procesos que cree primero el hijo izquierdo y derecho y posteriormente la rama izquierda crear 'n + 1' hijos, y la rama derecha constante de 3 hijos

A continuación observamos el diagrama de flujo del programa.



En el diagrama podemos observar que inicialmente se crean los dos hijos: izquierdo y derecho. Mediante una función recursiva se crean los hijos de los hijos de manera recursiva.

A continuación observamos la función:

```c

void hacerHijos(int niveles, int hijos, int hijo){
    if(niveles == 0)
        return;
    int i,j,estado;
    pid_t proceso;
    for(i = 0; i < hijos; i++){
        proceso = fork();

        switch( proceso ){
            case -1:
                printf("No se pudo crear el hijo\n", getpid());
                _exit(-1);
            case 0:
                printf("Soy el hijo %d mi padre es: %d\n", getpid(), getppid());
                if(hijo == 0){
                    hacerHijos(niveles-1, hijos + 1, 0);
                }
                else
                    hacerHijos(niveles-1, hijos, 1);
                _exit(0);
                break;
            default:
                if(i == (hijos - 1)){
                    for(j = 0; j < hijos; j++){
                        wait(&estado);
                    }
                }
                break;
        }
    }
}
```

Función: hacerHijos()

Parámetros: int niveles, int hijos, int hijo

Especificación:
niveles: el número de niveles que quieres crear
hijos: el número de hijos que quieres crear
hijo: 0 si se trata del hijo izquierdo o 1 si se trata del hijo derecho

Descripción:
La función crea el número de hijos que requiere con el padre a través de un contador.
Hay un contador principal para poder crear los procesos que se requieren y dentro de las funciones de los hijos están la recursividad en la función para poder volver a llamarla una vez que se creen los hijos.
Simultáneamente se van imprimiendo los pid's de los procesos.

Return:
La función no regresa ningún valor.

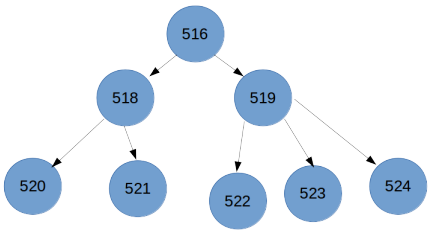
2.1 Ejemplo

En el siguiente ejemplo observaremos el programa corriendo con el valor de 3 niveles.
Utilizamos un árbol para vaciar el output y observar cómo se crea la estructura.

```
afront@afront-HP-Notebook:~/Documentos/SO/C/Practica 1/Código$ gcc -o proceso proceso.c
afront@afront-HP-Notebook:~/Documentos/SO/C/Practica 1/Código$ ./proceso
Introduce los niveles: 3
Soy el padre: 516
Soy el hijo izquierdo  518 mi padre es: 516
Soy el hijo 520 mi padre es: 518
Soy el hijo derecho 519 mi padre es: 516
Soy el hijo 521 mi padre es: 518
Soy el hijo 522 mi padre es: 519
Soy el hijo 523 mi padre es: 519
Soy el hijo 524 mi padre es: 519
afront@afront-HP-Notebook:~/Documentos/SO/C/Practica 1/Código$
```

Output del programa

Construyendo el árbol...



Árbol con la salida

3 Conclusiones

La desición de qué proceso se imprime primero sale de nuestras capacidades puesto que, el planificador es quien decide qué prioridad asignarlos por lo que no nos es posible administrar la forma en la que se imprimen los procesos hijos y padres.

A lo largo de la práctica existieron algunos problemas de desarrollo sobre todo para poder manera la manera en que los procesos se tienen que volver a ocupar para crear a sus hijos pero utilizando correctamente la función de *wait* y la funcion de *exit* logramos que estos se hicieran de manera recursiva y así se pudiesen crear todos.