Universidad Rey Juan Carlos

3º IST+TT-ADE

Software de Sistemas

Práctica 1 Llamadas al Sistema

Katia Leal

Procesos

Un proceso es un programa en ejecución, que debe ejecutar en forma secuencial y que tiene asociado un contexto. Este contexto se refiere a toda la información que describe completamente el estado de ejecución del proceso: los registros de la CPU, el contador de programa, pila (stack), los flags, etc. Para que en un sistema operativo puedan ejecutarse varias aplicaciones "a la vez" tenemos dos opciones: crear un proceso o crear un nuevo hilo de ejecución (thread). En realidad, salvo que tengamos varias CPU's, no se ejecutarán varias tareas a la vez. Será el sistema operativo el que asigne un tiempo de CPU a cada proceso, dando la impresión de que todos los procesos se estuvieran ejecutando en forma simultánea. Los procesos presentan alguna carencia en cuanto a eficiencia. Cuando tenemos varias instancias o procesos iguales, y el tiempo de CPU asignado a un proceso finaliza, se debe guardar en memoria todo el contexto de ese proceso. Así, cuando el planificador (o scheduler) del sistema operativo vuelva a asignar CPU a este proceso, se podría restaurar el estado anterior del proceso. Es lo que se denomina un cambio de contexto cuyo coste computacional es elevado.

Fundamentos de procesos

Escribe el siguiente código y guárdalo como procesos1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char *argv[])
{
      pid_t pid = 0;
      int i, n;
      int a = 1;
      if (argc != 2)
      {
            fprintf(stderr, "Uso: %s procesos\n", argv[0]);
            return 1;
      }
      n = atoi(argv[1]);
      for (i = 1; i < n; i++) {
            pid = fork();
            if (pid) {
                  a += i;
                  break;
            }
      }
      fprintf(stderr, "i:%d Proceso ID:%ld Padre ID:%ld Hijo ID:%ld
      a=%ld\n",
      i, (long)getpid(), (long)getppid(), (long)pid, a);
      return 0;
}
```

Compila el código mediante el comando:

```
$ gcc procesos1.c -o procesos1
```

Ejecuta con el siguiente comando:

```
$ ./procesos1 30
```

Cuestión 1: En la salida del programa, podrás observar que existen trazas que no están ordenadas según el orden en el que se crearon los procesos. ¿Cuál es la razón por la que se da este efecto?
Cuestión 2: ¿Qué añadirías para que cada proceso padre espere a su hijo antes de escribir por pantalla?
Cuestión 3: ¿Cuáles serán los valores de las variables 'a' e 'i' para cada proceso en el momento de su creación? ¿Por qué?
Cuestión 4: ¿Qué sucedería si al escribir la salida fprintf, lo hiciera en stdout en vez de stderr?
Cuestión 5: ¿Por qué puede fallar el fork? ¿Hay alguna forma de controlar el error?

IMPORTANTE: debe incluir todas las respuestas en el fichero de texto.

Creación y control de procesos
Siempre que entro en mi sesión en las X me gusta ejecutar una serie de programas, y tenerlos siempre funcionando y a la vista, por si necesito trabajar con ellos. Ir arrancando todos ellos, uno a uno, ya es suficientemente pesado; si además tengo que arrancar alguno de ellos otra vez si por alguna razón se muere, la tarea se vuelve tediosa.
En esta práctica te pedimos la implementación de un programa en el lenguaje C sobre Linux que me ayude en esa tarea. Debe hacer lo siguiente:

- Al principio, tiene que crear 5 hijos.
- Cada uno de esos cinco hijos debe ejecutar un programa diferente. Estos programas y sus argumentos serían los siguientes (sin las comillas):
  - "xload"
  - "xeyes"
  - "xlogo"
  - "xcalc"
  - "xclock update 1"
- Durante toda su ejecución monitorizará a sus hijos. Si por alguna razón uno de ellos muere debe volver a crearlo y hacer que ejecute el mismo programa que estaba ejecutando anteriormente. De esta manera siempre debemos tener los mismo 5 programas ejecutándose simultáneamente.

Pistas: Las llamadas al sistema necesarias para realizar esta práctica son fork(), wait(), exec().