Introducción a Sistemas Operativos: Procesos

Clips xxx
Francisco J Ballesteros

1. Errores

En el programa que hemos hecho y en los siguientes vamos a hacer muchas llamadas al sistema. Bueno, en realidad muchas serán llamadas a la librería de C y otras serán llamadas al sistema. Nos da un poco igual (salvo por la sección del manual en que están documentadas, que será la 2 para llamadas al sistema y la 3 para llamadas a la librería de C). En cualquier caso, son llamadas para utilizar UNIX.

En muchos casos no habrá problema en la llamada que hagamos podrá hacer su trabajo. Pero en otros casos no será así. O bien habremos utilizado argumentos incorrectos en la llamada, o bien tendremos un problema de permisos o de otro tipo. Por ejemplo, cada proceso tiene un directorio de trabajo (como sabes) y podemos utilizar la llamada *chdir(2)* para cambiarlo. ¡Pero es posible que intentemos cambiar a un directorio que no existe!

Todas las llamadas devuelven un valor que, entre otras cosas, nos indicará al menos si han podido o no hacer su trabajo. Es responsabilidad nuestra comprobar dicho valor y actuar en consecuencia. Habitualmente el valor devuelto por una función suele ser un valor absurdo en el caso de errores. Si la función devuelve un puntero, devolverá `NULL`, si devuelve un entero positivo, devolverá `-1`, etc. La tradición en UNIX es que si el valor devuelto es sólo para indicar un error, suele devolverse `-1` en caso de error y `0` en caso de éxito. Las páginas de manual de cada función indican qué se devuelve en caso de error.

Cuando programes en C deberías seguir el mismo convenio. Todas tus funciones deberían informar al que las llama de si hubo algún error o no (salvo en aquellos casos en que nunca pueda producirse un error y en aquellos casos en que si se produce un error la función termine la ejecución de todo el programa).

Debes **siempre comprobar si hubo errores** en tus programas, en todas las llamadas. Si no lo haces, el programa será un poltergueist. ¿Y qué debería hacerse cuando hubo un error? No hay una respuesta correcta a esta pregunta. Depende de lo que estés programando y de qué función tenga el error. Simplemente piensa qué te gustaría que pasara si en una llamada sufres un error. Imagina que el programa lo has tomado prestado y piensa en lo que crees que debería hacer en cada caso.

Una vez has comprobado que una llamada no ha podido hacer su trabajo (¡eso es lo que significa que sufrió un error!, los errores no son magia), deberías ver a qué se debió el error. En UNIX, puedes utilizar la global `errno` para ver el código del error (un entero) y puedes utilizar *strerror(3)* para obtener un string correspondiente a dicho código.

Por ejemplo, este programa intenta cambiar de directorio:

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

int
main(int argc, char* argv[])
{
    if (chdir("foo") < 0){
        printf("errno is %d\n", errno);
        printf("err string is '%s'\n", strerror(errno));
        exit(1);
    }
    /* ... do other things ... */
    exit(0);
}
```

Suponiendo que el ejecutable es `err` y que no existe el directorio `foo`, cuando lo ejecutamos...

```
unix$ err
errno is 2
err string is 'No such file or directory'
unix$
```

En la práctica, deberíamos imprimir el nombre del programa, lo que intentábamos hacer cuando hemos sufrido el error, y cuál es la causa:

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

int
main(int argc, char* argv[])
{
    if (chdir("foo") < 0){
        fprintf(stderr, "%s: chdir: foo: %s\n", argv[0], strerror(errno));
        exit(1);
    }
    /* ... do other things ... */
    exit(0);
}
```

Si ahora ejecutamos el programa y este no puede cambiar su directorio, obtenemos un mensaje que ayudará a resolver el problema:

```
unix$ err
err: chdir: foo: No such file or directory
unix$
```

Esto es tan habitual, que la función *warn(3)* hace justo eso. Por ejemplo:

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <err.h>

int
main(int argc, char* argv[])
{
    if (chdir("foo") < 0){
        warn("chdir: %s", "foo");
        exit(1);
    }
    /* ... do other things ... */
    exit(0);
}
```

Y ahora tenemos...

```
unix$ err
err: chdir: foo: No such file or directory
unix$
```

Como verás, `warn` se usa de un modo similar a *printf(3)* e imprime un mensaje de error indicando el nombre del programa y el string de error correspondiente a `errno`. Si has leído la página de manual *warn(3)* (¡Cosa que deberías haber hecho en este punto!) habrás visto que la función *err(3)* es similar a `warn` pero llama a `exit` tras imprimir el mensaje con el estatus que le pasas en el primer argumento.

Un último comentario. No tiene sentido utilizar `errno` si no es justo después de una llamada que ha fallado (y ha indicado su error con el valor de retorno). Sólo cuando una llamada a UNIX tiene un problema, dicha llamada actualiza `errno` para informar de la causa del error.