Tratamiento de errores en C. Juan Carlos Fernández Caballero, jfcaballero@uco.es

---

En el tratamiento de errores hay ocasiones en las que se puede utilizar un esquema como el siguiente. Como ejemplo se van a utilizar algunas funciones definidas en el estandar POSIX sobre procesos e hilos, pero se puede aplicar a cualquier otra función.

```
childpid = fork();

if (childpid == -1)
{
  perror("fork error");
  printf("errno value= %d\n", errno);
  exit(EXIT_FAILURE);
}
else
{
   ...
}
```

En este caso concreto, la documentación de OpenGroup dice lo siguiente: "Upon successful completion, fork() shall return 0 to the child process and shall return the process ID of the child process to the parent process....Otherwise, -1 shall be returned to the parent process, no child process shall be created, and errno shall be set to indicate the error."

Esto quiere decir que en caso de que la llamada tenga éxito se devuelve 0 al hijo y el ID del hijo al padre, pero en caso de error se devuelve -1 y se establece el código de error en la variable de entorno "errno". En este caso concreto de fork(), el codigo de error en "errno" se puede examinar mediante las macros [EAGAIN] y [ENOMEM].

Por tanto, siguiendo lo que especifica Posix, tal y como se muestra el control de errores anterior sobre la función fork(), se puede decir que NO es un control de errores completo (aunque si aceptable), ya que no se está consultado el valor posible de "errno", solo se muestra.

El siguiente código se utiliza "errno" cuando la función waitpid() devuelve -1. OpenGroup dice que el valor de "errno" puede ser el que se define en las siguiente macros:

[ECHILD]
    The process specified by pid does not exist or is not a child of the calling process, or the process group specified by pid does not exist or does not have any member process that is a child of the calling process.

[EINTR]
    The function was interrupted by a signal. The value of the location pointed to by stat_loc is undefined.

[EINVAL]
    The options argument is not valid.

Si hacemos uso de la especificación anterior, un ejemplo de uso de la macro [ECHILD] consultando "errno" tras la funcion waitpid() podría ser el siguiente:

```
hijo_pid=waitpid(-1, &estado_hijo, WUNTRACED | WCONTINUED);

while(hijo_pid !=-1)
{
    printf("Proceso ..: %d ", hijo_pid);
    //Comprueba el cambio de estado del proceso hijo.
    if (WIFEXITED(estado_hijo))
    {
        //Finalizó de forma normal. Muestra el valor que retornó.
        printf("Finalizó y devolvió el valor ..: %d )\n",WEXITSTATUS(estado_hijo));
    }
    else if (WIFSIGNALED(estado_hijo))
    {
        //Finalizó al recibir una señal no controlada.
        printf("Finalizó al recibir la señal ..: %d )\n", WTERMSIG(estado_hijo));
    }
    else if (WIFSTOPPED(estado_hijo))
    {
        //El proceso fue parado.
        printf("Parado ..: %d )\n", WSTOPSIG(estado_hijo));
```

```
    }
    else if (WIFCONTINUED(estado_hijo))
    {
        //El proceso fue reanudado.
        printf("Reanudado. )\n");
    }
    hijo_pid=waitpid(-1, &estado_hijo, WUNTRACED | WCONTINUED);
}

//Comprobamos el valor de errno
if(errno==ECHILD)
{
    //Ya no quedan más hijos por finalizar. Finalización correcta.
    printf ("Fin ejecución ID ..: %d (No quedan más hijos que finalizar.)\n", getpid());
}
if (errno==EINTR)
{
    ...
}
if (errno==EINVAL)
{
    ...
}
```

Por otro lado, existen funciones que vienen documentadas de la siguiente forma, como por ejemplo el caso de pthread_mutex_lock() (hay muchas más...):
"If successful, the pthread_mutex_lock(), pthread_mutex_trylock(), and pthread_mutex_unlock() functions shall return zero; otherwise, an error number shall be returned to indicate the error."

Aquí se dice que si la llamada no tiene éxito se devuelve un 0, y en caso contrario se devuelve un número que indica el tipo de error. Es decir, no os está devolviendo un número fijo de error como por ejemplo -1 y estableciendo el tipo en "errno" para que se consulte, sino que directamente devuelve un valor que hay que interpretar según la documentación.

Esto es así porque "errno" está definido como parte de la librería estándar de C. Si hay funciones que no forman parte de la librería estándar no tienen porque establecer el valor de error en "errno", y de ahí que se tenga que examinar su tipo de error. En la documentación de OpenGroup se indica explícitamente cuando una función establece un valor en "errno" o cuando devuelve un valor que debemos examinar usando macros.

Ejemplo para el caso concreto de pthread_mutex_lock():

```
error=pthread_mutex_lock(&mutex1);

switch (error)
{
  case EAGAIN: //Se ha excedido el máximo número de cierres recursivos.
  printf(" Se produjo el error EAGAIN (Se ha excedido el máximo número de cierres recursivos).\n");
  break;

  case EINVAL: //La hebra actual tiene mayor prioridad que el semáforo.
  printf(" Se produjo el error EINVAL (La hebra actual tiene mayor prioridad que el semáforo).\n");
  break;

  case ENOTRECOVERABLE: //No se puede recuperar el estado del semáforo.
  printf(" Se produjo el error ENOTRECOVERABLE (No se puede recuperar el estado del semáforo).\n");
  break;

  case EBUSY: //Semáforo cerrado.
  printf(" Se produjo el error EBUSY (Semáforo cerrado).\n");
  break;

  case EDEADLOCK: //Interbloqueo.
  printf(" Se produjo el error EDEADLOCK (Interbloqueo).\n");
  break;

  default: //Otro error no controlado.
  printf(" Se produjo el error %d\n", error);
  break;
}
```

pthread_mutex_lock() devuelve valores que están definidos en macros, pero perfectamente una función puede devolver un entero a interpretar y que no esté definido en una macro.