Función open / close

(https://pasky.wordpress.com/2009/08/04/funciones-openclose-y-readwrite-en-c/)

La función ***open*** devuelve un número entero que identifica a un descriptor de fichero o -1 en caso de error, y tiene como parámetros un puntero a la ruta del fichero que queramos abrir y unas banderas (flags) que indican la forma de apertura:

- sólo lectura (O_RDONLY)
- sólo escritura (O_WRONLY)
- lectura/escritura (O_RDWR)
- (O_CREAT)
- (O_EXCL)
- (O_NOCTTY)
- (O_TRUNC)
- (O_APPEND)
- (O_NONBLOCK)
- (O_NDELAY)
- (O_SYNC)
- (O_NOFOLLOW)
- (O_DIRECTORY)
- (O_DIRECT)
- (O_ASYNC)
- (O_LARGEFILE)

En la función *open* podemos especificar un tercer parámetro, el modo, que se usa para especificar los permisos en caso de que se esté creando el archivo. Dichos permisos o modos pueden ser:

- S_IRWXU (el usuario puede leer, escribir y ejecutar (rwx))
- S_IRUSR (el usuario puede leer (r))
- S_IWUSR (el usuario puede escribir (w))
- S_IXUSR (el usuario puede ejecutar (x))
- S_IRWXG
- S_IRGRP
- S_IWGRP
- S_IXGRP
- S_IRWXO
- S_IROTH
- S_IWOTH
- S_IXOTH.

(http://labsopa.dis.ulpgc.es/prog_c/FICHER.HTM#Heading4)

La función **open** abre un fichero ya existente, retornando un descriptor de fichero. La función tiene este prototipo:

```
int open ( char* nombre, int modo, int permisos );
```

El parámetro **nombre** es la cadena conteniendo el nombre del fichero que se quiere abrir.

El parámetro **modo** establece la forma en que se va a trabajar con el fichero. Algunas constantes que definen los modos básicos son:

O_RDONLY	abre en modo lectura
O_WRONLY	abre en modo escritura
O_RDWR	abre en modo lectura-escritura
O_APPEND	abre en modo apéndice (escritura desde el final)
O_CREAT	crea el fichero y lo abre (si existía, se lo machaca)
O_EXCL	usado con **O_CREAT**. Si el fichero existe, se retorna un error
O_TRUNC	abre el fichero y trunca su longitud a 0

Para usar estas constantes, han de incluir la cabecera **<fcntl.h>**. Los modos pueden combinarse, simplemente sumando las constantes, o haciendo un "or" lógico, como en este ejemplo:

```
O_CREAT | O_WRONLY
```

El parámetro ***acceso*** sólo se ha de emplear cuando se incluya la opción **O_CREAT**, y es un entero que define los permisos de acceso al fichero creado. Consulten en la bibliografía cómo se codifican los permisos.

La función **open** retorna un descriptor válido si el fichero se ha podido abrir, y el valor -1 en caso de error.

Función close

La función *close* cierra el descriptor de fichero que le pasemos como parámetro. Devuelve 0 en caso de éxito y -1 en caso de error.

Función Lectura / Escritura

Para leer y escribir información en ficheros, han de abrirlos primero con **`open`** o **`creat`**. Las funciones **`read`** y **`write`** se encargan de leer y de escribir, respectivamente:

```
int read ( int fichero, void* buffer, unsigned bytes );

int write( int fichero, void* buffer, unsigned bytes );
```

Ambas funciones toman un primer parámetro, ***fichero***, que es el descriptor del fichero sobre el que se pretende actuar.

El parámetro ***buffer*** es un apuntador al área de memoria donde se va a efectuar la transferencia. O sea, de donde se van a leer los datos en la función ***read***, o donde se van a depositar en el caso de ***write***.

El parámetro ***bytes*** especifica el número de bytes que se van a transferir.

Las dos funciones devuelven el número de bytes que realmente se han transferido. Este dato es particularmente útil en la función ***read***, pues es una pista para saber si se ha llegado al final del fichero. En caso de error, retornan un -1.

Hay que tener especial cautela con estas funciones, pues el programa no se va a detener en caso de error, ni hay control sobre si el puntero ***buffer*** apunta a un área con capacidad suficiente (en el caso de la escritura), etc., etc.

La primera vez que se lee o escribe en un fichero recién abierto, se hace desde el principio del fichero (desde el final si se incluyó la opción **`O_APPEND`**). El puntero del fichero se mueve al *byte* siguiente al último *byte* leído o escrito en el fichero. Es decir, UNIX trabaja con ficheros secuenciales.