Nivel 10: **directorios.c** {mi_link(), mi_unlink()}, **mi_link.c**, **mi_rm.c**, [**mi_rmdir**] y **scripts**

comandos:	**directorios.c:**	**ficheros.c:**
mi_mkdir, [mi_touch]	mi_creat()	
mi_ls	mi_dir()	
mi_chmod	mi_chmod()	mi_chmod_f()
mi_stat	mi_stat()	mi_stat_f()
mi_escribir	mi_write()	mi_write_f()
mi_cat	mi_read()	mi_read_f()
mi_link	**mi_link()**	
mi_rm, [mi_rmdir], [mi_rm_r]	**mi_unlink()**	**mi_truncar_f()**

Tabla con la correlación de comandos y funciones de la capa de directorios y la capa de ficheros

7) Creación de enlaces físicos[1]

7a) mi_link.c

Sintaxis: **`./mi_link disco /ruta_fichero_original /ruta_enlace`**

Programa **mi_link.c** que crea un enlace a un **fichero**, llamando a la función mi_link() de la capa de directorios.

Observaciones:

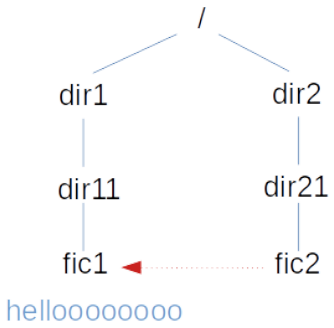
- Hay que comprobar que las sintaxis de las rutas se correspondan a un **fichero** ya que no haremos enlaces de directorios.
- ruta_fichero_original **ha de existir** y ruta_enlace **NO ha de existir** (eso lo comprobará mi_link() de la capa de directorios)

Podéis probar el **comando link o ln[2] de Linux** con los siguientes ejemplos:

[1] Diferencia entre enlace físico (hard link) y enlace simbólico (soft link). La información que guarda el **enlace simbólico** es el nombre del archivo enlazado, si el archivo enlazado cambia de nombre, el enlace simbólico automáticamente queda roto. Un **enlace físico** ("permanente" o "duro") no presenta ese problema debido a que el inodo tiene el mismo valor, se pueden cambiar los nombres de ambos ficheros y la relación se mantiene.

[2] **ln** tiene más opciones que **link** y sirve tanto para crear enlaces físicos como simbólicos (**ln -s**)

```
uib:~$ mkdir dir1
uib:~$ mkdir dir1/dir11/
uib:~$ cat > dir1/dir11/fic1 #camino1
hellooooooo
uib:~$ mkdir dir2
uib:~$ mkdir dir2/dir21/
uib:~$ ln dir1/dir11/fic1 dir2/dir21/fic2 #o comando link
```




```
uib:~$ cat dir2/dir21/fic2 #ha de mostrar mismo contenido que dir1/dir11/fic1
hellooooooo
uib:~$ ls -i dir1/dir11/fic1 #la opción -i muestra nº inodo
675745 dir1/dir11/fic1
uib:~$ ls -i dir2/dir21/fic2 #comprobamos nº inodo
675745 dir2/dir21/fic2
uib:~$ stat dir1/dir11/fic1 #comprobamos nº de enlaces
  Fichero: dir1/dir11/fic1
  Tamaño: 11            Bloques: 8          Bloque E/S: 4096   fichero regular
Dispositivo: 802h/2050d   Nodo-i: 675745     Enlaces: 2
Acceso: (0664/-rw-rw-r--)  Uid: ( 1000/     uib)   Gid: ( 1000/     uib)
Acceso: 2021-05-03 17:55:25.991083965 +0200
Modificación: 2021-05-03 17:53:56.418393504 +0200
     Cambio: 2021-05-03 17:54:50.806812743 +0200
    Creación: -
```

```
uib:~$ ln dir1/dir11/fic3 dir2/dir21/fic4 #camino1 ha de existir
ln: fallo al acceder a 'dir1/dir11/fic3': No existe el archivo o el directorio
uib:~$ touch dir1/dir11/fic3
uib:~$ ln dir1/dir11/fic3 dir2/dir21/fic4
uib:~$ ln dir1/dir11/fic3 dir2/dir21/fic5
uib:~$ stat dir1/dir11/fic3
  Fichero: dir1/dir11/fic3
  Tamaño: 0             Bloques: 0          Bloque E/S: 4096   fichero regular vacío
Dispositivo: 802h/2050d   Nodo-i: 683007     Enlaces: 3
Acceso: (0664/-rw-rw-r--)  Uid: ( 1000/     uib)   Gid: ( 1000/     uib)
Acceso: 2021-05-03 17:58:45.996625860 +0200
```

```
Modificación: 2021-05-03 17:58:45.996625860 +0200
     Cambio: 2021-05-03 17:59:18.780878621 +0200
   Creación: -
uib:~$ ln dir1/dir11/fic3 dir2/dir21/fic2 #camino2 NO ha de existir
ln: fallo al crear el enlace duro 'dir2/dir21/fic2': El archivo ya existe
```

7b) int **mi_link**(const char *camino1, const char *camino2);

Crea el enlace de una entrada de directorio camino2 al inodo especificado por otra entrada de directorio camino1 .

Hay que comprobar que la **entrada** camino1 **exista**. Obtener el nº de inodo asociado, p_inodo1, mediante la función buscar_entrada() y comprobar que tiene **permiso de lectura**.

camino1 y camino2 han de referirse a un fichero!!! [3]

En el caso de que la entrada de camino2 no exista, la creamos mediante la función buscar_entrada() con permisos 6 (la podemos llamar directamente en modalidad escritura y que nos devuelva error en caso de que **la entrada ya exista**).
Si la entrada se ha creado correctamente entonces:

- Leemos la entrada creada correspondiente a camino2, o sea la entrada p_entrada2 de p_inodo_dir2
- Creamos el enlace: Asociamos a esta entrada **el mismo inodo** que el asociado a la entrada de camino1, es decir p_inodo1.
- Escribimos la entrada modificada en p_inodo_dir2
- Liberamos el inodo que se ha asociado a la entrada creada, p_inodo2
- Incrementamos la cantidad de enlaces de p_inodo1, actualizamos el ctime y lo salvamos

TESTS DE PRUEBA

```
uib:~$ ./test10.sh
##################################################################
                        1ª parte
##################################################################
$ ./mi_mkfs disco 100000
##################################################################
```

[3] No se permite el enlace a directorios para evitar que se creen ciclos en el grafo.

```
$ ./mi_mkdir disco 6 /dir1/
$ ./mi_mkdir disco 6 /dir1/dir11/
$ ./mi_touch disco 6 /dir1/dir11/fic1
$ ./mi_escribir disco /dir1/dir11/fic1 helloooooo 0
longitud texto: 11
Bytes escritos: 11
########################################################
$ ./mi_mkdir disco 6 /dir2/
$ ./mi_mkdir disco 6 /dir2/dir21/
########################################################
$ ./mi_link disco /dir1/dir11/fic1 /dir2/dir21/fic2
########################################################
$ ./mi_cat disco /dir2/dir21/fic2 #ha de mostrar mismo contenido que /dir1/dir11/fic1
helloooooo

Total_leidos 11
$ ./mi_stat disco /dir1/dir11/fic1
Nº de inodo: 3
tipo: f
permisos: 6
atime: Thu 2021-05-06 11:26:32
ctime: Thu 2021-05-06 11:26:32
mtime: Thu 2021-05-06 11:26:32
nlinks: 2
tamEnBytesLog: 11
numBloquesOcupados: 1

$ ./mi_stat disco /dir2/dir21/fic2
Nº de inodo: 3
tipo: f
permisos: 6
atime: Thu 2021-05-06 11:26:32
ctime: Thu 2021-05-06 11:26:32
mtime: Thu 2021-05-06 11:26:32
nlinks: 2
tamEnBytesLog: 11
numBloquesOcupados: 1

########################################################
$ ./mi_link disco /dir1/dir11/fic3 /di2/dir21/fic4  #camino1 ha de existir
Error: No existe el archivo o el directorio.
$ ./mi_touch disco 6 /dir1/dir11/fic3
$ ./mi_link disco /dir1/dir11/fic3 /dir2/dir21/fic4
$ ./mi_link disco /dir1/dir11/fic3 /dir2/dir21/fic5
$ ./mi_stat disco /dir1/dir11/fic3
```

```
Nº de inodo: 6
tipo: f
permisos: 6
atime: Thu 2021-05-06 11:26:32
ctime: Thu 2021-05-06 11:26:32
mtime: Thu 2021-05-06 11:26:32
nlinks: 3
tamEnBytesLog: 0
numBloquesOcupados: 0

############################################################
$ ./mi_link disco /dir1/dir11/fic3 /dir2/dir21/fic2 #camino2 NO ha de existir
Error: El archivo ya existe.
```

8) Borrado de enlaces, ficheros y directorios

8a) mi_rm.c

Sintaxis: `./mi_rm disco /ruta`

Programa **mi_rm.c** que borra un fichero o directorio, llamando a la función mi_unlink() de la capa de directorios.

Observaciones:

- No se ha de poder borrar el **directorio raíz**.
  - La función mi_unlink() de la capa de directorios ha de comprobar que si se trata de un **directorio ha de estar vacío** para poder borrarlo.
  - Opcionalmente se puede hacer que **mi_rm** sea sólo para borrar un fichero y crear un comando adicional **mi_rmdir** para borrar un directorio.
  - También se podría crear otro programa adicional llamado por ejemplo **mi_rm_r.c** (r de recursivo) que borrase todo el contenido de un **directorio no vacío** (similar al comando **rm** con la opción **-r** de Linux), o admitir un parámetro que indicase esta opción.

Podéis probar los **comandos rm y rmdir de Linux** con el siguiente ejemplo (continuación del anterior):

```
uib:~$ rmdir dir2/dir21
rmdir: fallo al borrar 'dir2/dir21': El directorio no está vacío
uib:~$ rm dir2/dir21/fic2
uib:~$ stat dir1/dir11/fic1 #hemos borrado 1 enlace
  Ficher: 12            Bloques: 8        Bloque E/S: 4096   fichero regular
```

```
Dispositivo: 802h/2050d    Nodo-i: 675745     Enlaces: 1
Acceso: (0664/-rw-rw-r--)  Uid: ( 1000/     uib)   Gid: ( 1000/     uib)
Acceso: 2021-05-11 12:46:04.949073235 +0200
Modificación: 2021-05-11 12:45:11.469099514 +0200
     Cambio: 2021-05-11 13:43:28.360606445 +0200
   Creación: -
uib:~$ rm dir2/dir21/fic2
rm: no se puede borrar 'dir2/dir21/fic2': No existe el archivo o el directorio
uib:~$ rmdir dir2/dri21
rmdir: fallo al borrar 'dir2/dri21': No existe el archivo o el directorio
uib:~$ ls -l dir2/dir21
total 0
-rw-rw-r-- 3 uib uib 0 de maig  11 13:37 fic4
-rw-rw-r-- 3 uib uib 0 de maig  11 13:37 fic5
uib:~$ rm dir2/dir21/fic4
uib:~$ rm dir2/dir21/fic5
uib:~$ rmdir dir2/dir21
uib:~$ ls -l dir2
total 0
```

8b) int mi_unlink(const char *camino);

Función de la capa de directorios que borra la entrada de directorio especificada (no hay que olvidar actualizar la cantidad de enlaces en el inodo) y, en caso de que fuera el último enlace existente, borrar el propio fichero/directorio.
Es decir que esta función nos servirá tanto para borrar un enlace a un fichero como para eliminar un fichero o directorio que no contenga enlaces.

Cuando queramos borrar por ejemplo /fic1, mi_unlink():

- En primer lugar **eliminamos la entrada de directorio** fic1 del directorio raíz y **decrementamos** nlinks.
- Si nlinks=0 entonces liberamos el inodo y todo su contenido con liberar_inodo(), la cual llamará a liberar_bloques_inodo() para liberar tanto los bloques de datos como de punteros del fichero.

Paso a paso:

- Hay que comprobar que la **entrada camino exista** y obtener su nº de entrada (p_entrada), mediante la función buscar_entrada().
- Si se trata de un **directorio** y **no está vacío** (inodo.tamEnBytesLog > 0) entonces **no se puede borrar** y salimos de la función. En caso contrario:

- Mediante la función leer_inodo() leemos el inodo asociado al directorio que contiene la entrada que queremos eliminar (p_inodo_dir), y obtenemos el nº de entradas que tiene (inodo_dir.tamEnBytesLog/sizeof(struct entrada)).
- Si la entrada a eliminar es la última (p_entrada ==nº entradas - 1), basta con **truncar el inodo** a su tamaño menos el tamaño de una entrada, mediante la función mi_truncar_f().
- Si no es la última entrada, entonces tenemos que leer la última y colocarla en la posición de la entrada que queremos eliminar (p_entrada), y después ya podemos truncar el inodo como en el caso anterior. De esta manera siempre dejaremos las entradas de un directorio **consecutivas (sin huecos)** para cuando tengamos que utilizar la función buscar_entrada()
- Leemos el inodo asociado a la entrada eliminada para **decrementar el nº de enlaces.**
- **Si no quedan enlaces (nlinks) entonces liberaremos el inodo**, en caso contrario actualizamos su ctime y escribimos el inodo.

TEST DE PRUEBA (ejecución continuación del anterior)

```
##################################################################
                        2ª parte
##################################################################
$ ./mi_rm disco /dir2/dir21/ #o mi_rmdir
Error: El directorio /dir2/dir21/ no está vacío
$ ./mi_rm disco /dir2/dir21/fic2
liberados: 0
$ ./mi_stat disco /dir1/dir11/fic1 #Hemos borrado 1 enlace
Nº de inodo: 3
tipo: f
permisos: 6
atime: Thu 2021-05-06 11:36:51
ctime: Thu 2021-05-06 11:36:51
mtime: Thu 2021-05-06 11:36:51
nlinks: 1
tamEnBytesLog: 11
numBloquesOcupados: 1

$ ./mi_rm disco /dir2/dir21/fic2
Error: No existe el archivo o el directorio.
$ ./mi_rm disco /dir2/dir21/ #o mi_rmdir
Error: El directorio /dir2/dir21/ no está vacío
$ ./mi_ls disco /dir2/dir21/
```

```
Total: 2
Tipo    Modo  mTime                          Tamaño                 Nombre
------------------------------------------------------------------------------
f       rw-   2021-05-06 11:36:51            0                      fic5
f       rw-   2021-05-06 11:36:51            0                      fic4

$ ./mi_rm disco /dir2/dir21/fic4
liberados: 0
$ ./mi_rm disco /dir2/dir21/fic5
liberados: 1
$ ./mi_rm disco /dir2/dir21/ #o mi_rmdir
liberados: 1
$ ./mi_ls disco /dir2/
Total: 0

##################################################################
```

```
##################################################################
Comprobamos que al crear 17 subdirectorios los bloques de datos del padre son 2
(en un bloque caben 16 entradas de directorio),
 y que al eliminar un subdirectorio el directorio padre tiene 1 bloque de datos
##################################################################
$ ./mi_mkdir disco /d1/
##################################################################
creamos 17 subdirectorios sd0, sd1..., sd16 en d1
$ for i in $(seq 0 16)
> do
> ./mi_mkdir disco 6 /d1/sd$i/
> done
##################################################################
Mostramos la metainformacion del directorio para ver que tiene 2 bloques de datos
$ ./mi_stat disco /d1/
Nº de inodo: 5
tipo: d
permisos: 6
atime: Thu 2021-05-06 11:36:51
ctime: Thu 2021-05-06 11:36:51
mtime: Thu 2021-05-06 11:36:51
nlinks: 1
tamEnBytesLog: 1088
numBloquesOcupados: 2

##################################################################
Listamos el directorio para ver sus subdirectorios
$ ./mi_ls disco /d1/
```

```
Total: 17
Tipo    Modo    mTime                   Tamaño              Nombre
------------------------------------------------------------------------------
d       rw-     2021-05-06 11:36:51     0                   sd0
d       rw-     2021-05-06 11:36:51     0                   sd1
d       rw-     2021-05-06 11:36:51     0                   sd2
d       rw-     2021-05-06 11:36:51     0                   sd3
d       rw-     2021-05-06 11:36:51     0                   sd4
d       rw-     2021-05-06 11:36:51     0                   sd5
d       rw-     2021-05-06 11:36:51     0                   sd6
d       rw-     2021-05-06 11:36:51     0                   sd7
d       rw-     2021-05-06 11:36:51     0                   sd8
d       rw-     2021-05-06 11:36:51     0                   sd9
d       rw-     2021-05-06 11:36:51     0                   sd10
d       rw-     2021-05-06 11:36:51     0                   sd11
d       rw-     2021-05-06 11:36:51     0                   sd12
d       rw-     2021-05-06 11:36:51     0                   sd13
d       rw-     2021-05-06 11:36:51     0                   sd14
d       rw-     2021-05-06 11:36:51     0                   sd15
d       rw-     2021-05-06 11:36:51     0                   sd16

############################################################
Eliminamos el subdirectorio sd3 de d1
$ ./mi_rm disco /d1/sd3/ #o mi_rmdir
liberados: 1
############################################################
Mostramos la metainformacion de d1 para ver que ahora tiene 1 bloque de datos
$ ./mi_stat disco /d1/
Nº de inodo: 5
tipo: d
permisos: 6
atime: Thu 2021-05-06 11:36:51
ctime: Thu 2021-05-06 11:36:51
mtime: Thu 2021-05-06 11:36:51
nlinks: 1
tamEnBytesLog: 1024
numBloquesOcupados: 1

############################################################
Volvemos a listar el directorio para ver que se ha eliminado un subdirectorio
$ ./mi_ls disco /d1/
Total: 16
Tipo    Modo    mTime                   Tamaño              Nombre
------------------------------------------------------------------------------
d       rw-     2021-05-06 11:36:51     0                   sd0
```

d	rw-	2021-05-06 11:36:51	0	sd1
d	rw-	2021-05-06 11:36:51	0	sd2
d	rw-	2021-05-06 11:36:51	0	sd16
d	rw-	2021-05-06 11:36:51	0	sd4
d	rw-	2021-05-06 11:36:51	0	sd5
d	rw-	2021-05-06 11:36:51	0	sd6
d	rw-	2021-05-06 11:36:51	0	sd7
d	rw-	2021-05-06 11:36:51	0	sd8
d	rw-	2021-05-06 11:36:51	0	sd9
d	rw-	2021-05-06 11:36:51	0	sd10
d	rw-	2021-05-06 11:36:51	0	sd11
d	rw-	2021-05-06 11:36:51	0	sd12
d	rw-	2021-05-06 11:36:51	0	sd13
d	rw-	2021-05-06 11:36:51	0	sd14
d	rw-	2021-05-06 11:36:51	0	sd15

Otras funcionalidades extras

Podéis crear **otras funcionalidades adicionales** para el sistema de ficheros como por ejemplo renombrar un fichero/directorio, mover un fichero/directorio, copiar un fichero/directorio, … que podrán ser entregadas también después de la revisión final.