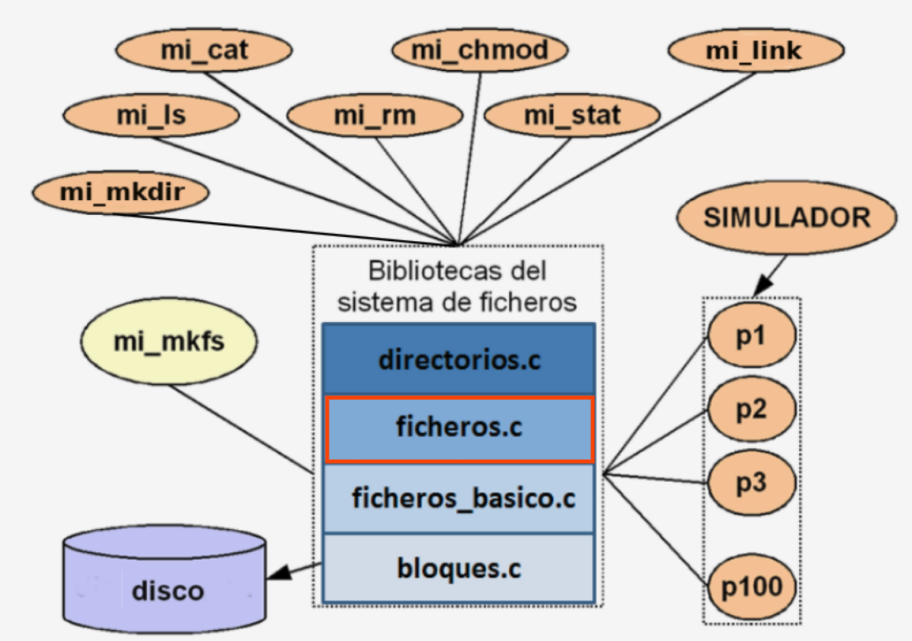
Nivel 5: ficheros.c {mi_write_f(), mi_read_f(), mi_chmod_f(), mi_stat_f()} y escribir.c, leer.c, permitir.c

Empezaremos a construir el programa **ficheros.c** de nuestra biblioteca de funciones (en este punto un fichero sigue identificado únicamente por el **número de su inodo**, es decir por su posición en el array de inodos).



En este nivel desarrollaremos las funciones que nos permitan escribir, *mi_write_f()*, y leer, *mi_read_f()*, cierta cantidad de bytes, *nbytes*, en un fichero (identificado por su nº de inodo, *ninodo*), a partir de una posición lógica cualquiera del fichero expresada en bytes, *offset*. Indicaremos con un puntero de tipo *void* la dirección del contenedor, *buf_original*, donde almacenaremos lo que leamos del fichero o desde donde volcaremos al fichero lo que escribamos, de esta forma el contenedor podrá contener cualquier tipo de datos.
Además implementaremos una función, *mi_stat_f()*, para obtener los metadatos del inodo (todos los campos menos los punteros), y otra función, *mi_chmod_f()* para cambiar los permisos de un inodo.

1) int mi_write_f(unsigned int *ninodo*, const void **buf_original*, unsigned int *offset*, unsigned int *nbytes*);

Escribe el contenido procedente de un buffer de memoria, *buf_original*, de tamaño *nbytes*, en un fichero/directorio (correspondiente al inodo pasado como argumento, *ninodo*): le

indicamos la posición de escritura inicial en bytes lógicos, *offset*, con respecto al inodo, y el número de bytes, *nbytes*, que hay que escribir.
Hay que devolver la cantidad de bytes escritos realmente (si todo ha ido bien coincidirá con *nbytes* pero podría haberse producido un error en alguna operación de escritura física en el dispositivo).

Esta operación sólo está permitida cuando haya **permiso de escritura** sobre el inodo (opción 'w' a 1), es decir que permisos tenga el valor 010, 011, 110 o 111, lo cual se puede averiguar con la siguiente comparación:

```
if ((inodo.permisos & 2) != 2) ...
```

Necesitamos saber de qué bloque a qué bloque lógico hay que escribir:

- Calculamos cuál va a ser el primer bloque lógico, *pimerBL*, donde hay que escribir:
  ```
  primerBL = offset / BLOCKSIZE;
  ```
- Calculamos cuál va a ser el último bloque lógico, *ultimoBL*, donde hay que escribir:
  ```
  ultimoBL = (offset + nbytes - 1) / BLOCKSIZE;
  ```

Y también los desplazamientos dentro de esos bloques donde cae el *offset*, y los *nbytes* escritos a partir del *offset*:

- Calculamos el desplazamiento *desp1* en el bloque para el *offset*:
  ```
  desp1 = offset % BLOCKSIZE;
  ```
- Calculamos el desplazamiento *desp2* en el bloque para ver donde llegan los *nbytes* escritos a partir del *offset*:
  ```
  desp2 = (offset + nbytes - 1) % BLOCKSIZE;
  ```

Primeramente trataremos el **caso en que el** (*primerBL*==*ultimoBL*), y por tanto el *buffer* que vamos a escribir, *buf_original*, cabe en un solo bloque.
Cualquier bloque del sistema de ficheros que no vaya a ser escrito en su totalidad, (mediante *bwrite()*) ha de ser previamente leído (mediante *bread()*) para preservar el valor de los bytes no escritos.

Pasos a seguir:

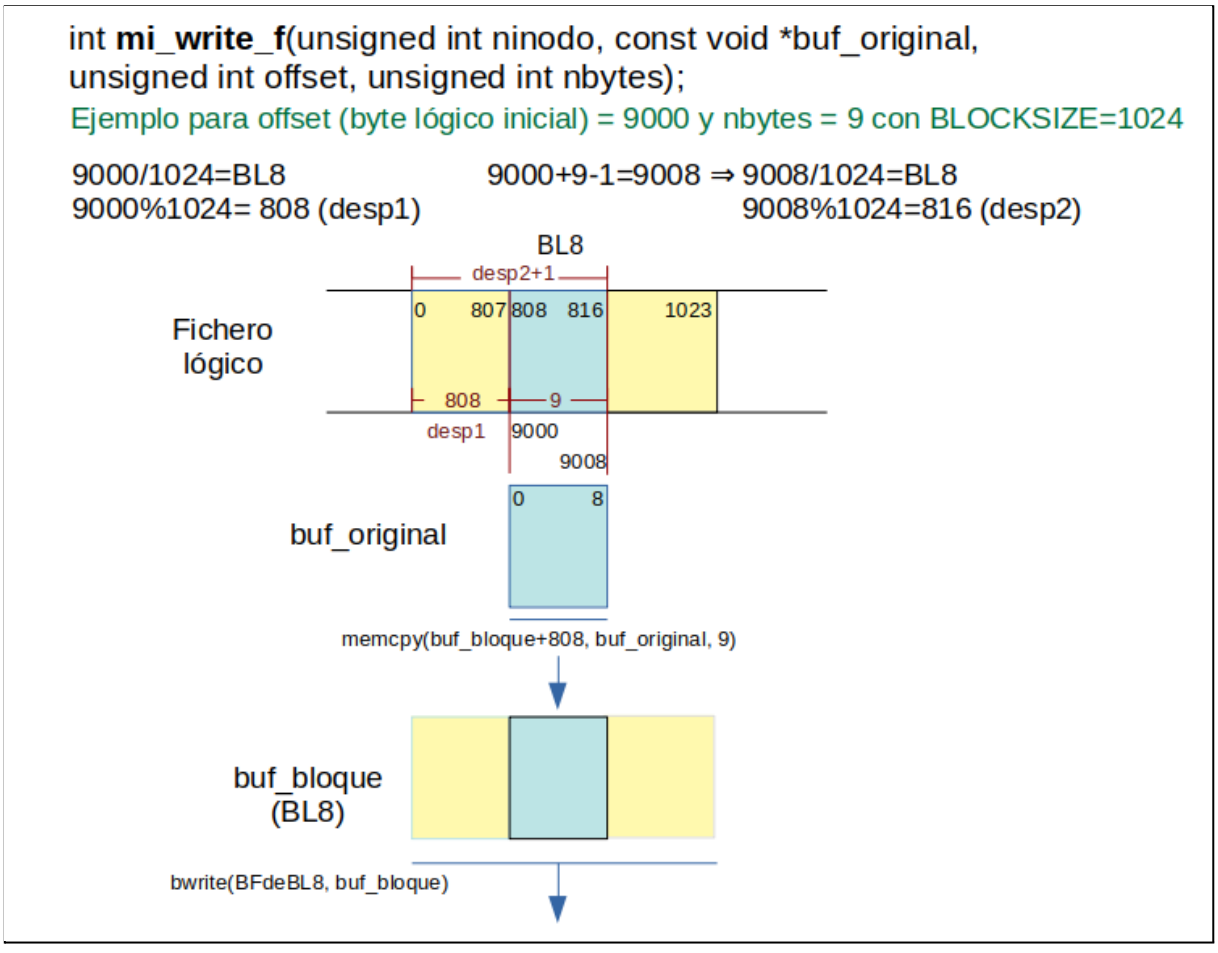
- Obtenemos el nº de bloque físico, *nbfisico*, correspondiente a *primerBL*, mediante *traducir_bloque_inodo()* (con *reservar* = 1).

- Leemos ese bloque físico del dispositivo virtual y lo almacenamos en un array de caracteres del tamaño de un bloque, *buf_bloque*.
- Utilizamos *memcpy()* para escribir los *nbytes* (siendo *nbytes* < *BLOCKSIZE*), o lo que es lo mismo *desp2* - *desp1* + *1* bytes, del *buf_original* en la posición *buf_bloque* + *desp1*:

```
memcpy(buf_bloque + desp1, buf_original, nbytes);
```

- Escribimos *buf_bloque* modificado en el nº de bloque físico correspondiente, *nbfisico*.

Para entenderlo mejor, veamos un ejemplo en el que queremos escribir 9 bytes (*nbytes*) (almacenados en el *buffer* *buf_original*), a partir del byte lógico número 9000 (*offset*) del inodo *ninodo*. Suponemos que el tamaño de bloque es de 1024 bytes:



- El primer byte lógico que vamos a escribir es el número 9000 (*offset*).
- Buscamos a qué bloque lógico pertenece, para ello calculamos 9000 / 1024 = 8 (primer bloque lógico, *primerBL*, donde vamos a escribir).
- El último byte lógico que vamos a escribir es el 9008 (*offset* + *nbytes* - *1*): 9000 + 9 – 1.

- Buscamos a qué bloque lógico pertenece (*ultimoBL*): Calculamos 9008 / 1024 = 8 (el último bloque lógico coincide con el primero).
- Utilizamos la función *traducir_bloque_inodo (ninodo, primerBL, 1)* para obtener el nº de bloque físico, *nbfisico*.
- Hacemos un *bread()* de tal bloque físico y almacenaremos el resultado de la lectura en un buffer de memoria principal llamado *buf_bloque* (de tamaño *BLOCKSIZE*), para poder preservar aquellos bytes que no se tengan que sobreescribir.
- Obtenemos el byte dentro del bloque lógico 8 donde cae el *offset* pasado por paŕametro, y que denominaremos *desp1*:

  9000 % 1024 = 808

  Es decir la escritura de los 9 bytes en el *offset* 9000 del fichero lógico se iniciará en el byte 808 del bloque lógico 8 de ese inodo.
- Calculamos el byte lógico hasta donde hemos de escribir, *desp2*:

  (9000+9-1) % 1024 = 816
- Por tanto tendremos que copiar *desp2* - *desp1* + 1 bytes (que es lo mismo que *nbytes*) desde el *bufer_original* al *buf_bloque* en la posición indicada por *desp1*:

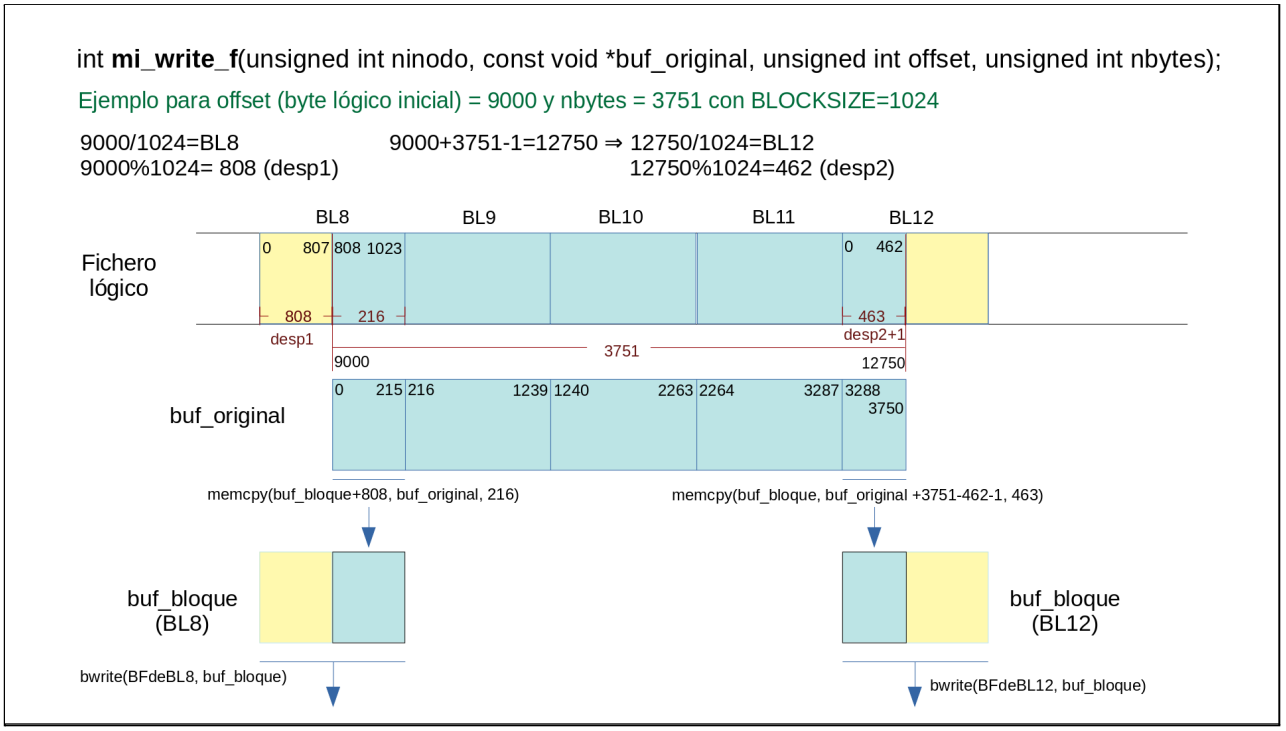
  memcpy (*buf_bloque* + 808, *buf_original*, 9)

- Obtenemos el *nbfisico* correspondiente al bloque lógico 8, utilizando *traducir_bloque_inodo()*.
- Hacemos un *bwrite() en ese nbfisico usando el contenedor buf_bloque* (con los nuevos datos escritos, preservando los que contenía).

A continuación trataremos el **caso en que la operación de escritura afecte a más de un bloque.**

Para entender mejor cómo ha de trabajar esta función, veamos un ejemplo en el que queremos escribir 3571 bytes (almacenados en el buffer *buf_original*) a partir del byte lógico número 9000 del inodo (*offset*). Suponemos que el tamaño de bloque es de 1024 bytes:

- El primer byte lógico que vamos a escribir (*offset*) es el número 9000.
- Buscamos a qué bloque lógico pertenece, para ello calculamos *primerBL* = 9000 / 1024 = 8.
- El último byte lógico, que vamos a escribir es el 12750: 9000 + 3751 – 1.
- Buscamos a qué bloque lógico pertenece: Calculamos *ultimoBL* = 12750 / 1024 = 12.



Distingamos tres fases:

1. **Primer bloque lógico** (BL8):

   Habrá que preservar una parte del contenido original que había en el dispositivo virtual y sobreescribir otra con el contenido correspondiente del *buffer_original*:

   - Hacemos un *bread()* de su bloque físico, *nbfisico*, obtenido con la función *traducir_bloque_inodo (ninodo, primerBLogico, 1)* y almacenamos el resultado en un buffer de memoria principal llamado *buf_bloque* (de tamaño *BLOCKSIZE*), para poder preservar aquellos bytes que no se tengan que sobreescribir.
   - Obtenemos el byte dentro del bloque lógico 8 donde cae el *offset* pasado por parámetro, y que denominaremos *desp1*: 9000 % 1024 = 808. Es decir la escritura de los 3751 bytes en el *offset* 9000 del fichero lógico se iniciará en el byte 808 del bloque lógico 8 de ese inodo.
   - Los restantes bytes, *BLOCKSIZE* - *desp1*, o sea 1024 - 808 = 216, son los que se han de copiar del *buf_original* al *buf_bloque* en la posición indicada por *desp1*:

     memcpy (*buf_bloque* + 808, *buf_original*, 216)

   - Hacemos un *bwrite()* del *buf_bloque* (con los nuevos datos, preservando los que contenía) en el bloque físico, *nbfisico*, que nos había devuelto *traducir_bloque_inodo()* para el bloque lógico 8.

2. **Bloques lógicos intermedios** (bloques 9, 10, 11):
   No hay que preservar datos ya que vamos a sobreescribir bloques completos, por tanto no hace falta leerlos previamente ni utilizar el *memcpy* sino escribir directamente en el dispositivo el bloque correspondiente del *buf_original*.
   - Iteramos para cada bloque lógico intermedio *i*, obteniendo el contenido a escribir del fragmento correspondiente del *buf_original*:

     ```
     buf_original + (BLOCKSIZE - desp1) + (i - primerBL - 1) * BLOCKSIZE
     ```

     y lo volcamos al dispositivo mediante un *bwrite()* en el bloque físico, *nbfisico*, correspondiente a ese bloque lógico. O sea:

     bwrite(*nbfisico*, *buf_original* + (1024 – 808) + (*i* – 8 – 1) * 1024)

3. **Último bloque lógico** (bloque 12):
   Habrá que sobreescribir con el último fragmento de *buf_original* y preservar la parte restante del contenido original que había en el dispositivo virtual:
   - Hacemos un *bread()* del bloque físico correspondiente, *nbfisico* (obtenido mediante la función *traducir_bloque_inodo(ninodo, ultimoBLogico, 1))* y almacenamos el resultado en el *buffer* *buf_bloque* de tamaño *BLOCKSIZE*.
   - Calculamos el byte lógico del último bloque hasta donde hemos de escribir, es decir el desplazamiento en el último bloque, *desp2*: 12750 % 1024 = 462
   - Copiamos esos bytes a *buf_bloque*:

     ```
     memcpy(buf_bloque, buf_original + (nbytes - desp2 - 1), desp2 + 1);
     ```

     O sea:

     memcpy (*buf_bloque*, *buf_original* + (3751 – 462 - 1), 462 + 1)

   - Hacemos un *bwrite()* del *buf_bloque* (que ahora contiene los nuevos datos, preservando los restantes originales) en la posición *nbfisico* correspondiente a ese bloque lógico.

Finalmente actualizaremos la metainformación del inodo:

- Leer el inodo actualizado.
- Actualizar el tamaño en bytes lógico, *tamEnBytesLog*, **solo si hemos escrito más allá del final del fichero**[1], y por tanto el *ctime* si modificamos cualquier

[1] El EOF nos lo da el tamaño en bytes lógico del inodo (que es el byte más alto escrito hasta el momento) y que guardamos como campo en el inodo. Como las escrituras que hacemos pueden ser con acceso directo

campo del inodo.
- Actualizar el *mtime* (porque hemos escrito en la zona de datos).
- Escribir el inodo.

Hay que devolver la cantidad de bytes escritos.

2) int mi_read_f(unsigned int *ninodo*, void **buf_original*, unsigned int *offset*, unsigned int *nbytes*);

Lee información de un fichero/directorio (correspondiente al nº de inodo, *ninodo*, pasado como argumento) y la almacena en un buffer de memoria, *buf_original*: le indicamos la posición de lectura inicial *offset* con respecto al inodo (en bytes) y el número de bytes *nbytes* que hay que leer.

Esta operación sólo está permitida cuando haya **permiso de lectura** sobre el inodo (opción 'r'), es decir que permisos tenga el valor 100, 101, 110 o 111, lo cual se puede averiguar con la siguiente comparación:

```
if ((inodo.permisos & 4) != 4) ...
```

La función no puede leer más allá del tamaño en bytes lógicos del inodo, *tamEnBytesLog* (es decir, más allá del EOF):

```
si offset >= inodo.tamEnBytesLog entonces
   leidos := 0 // No podemos leer nada
   devolver leidos
fsi
si (offset + nbytes) >= inodo.tamEnBytesLog // pretende leer más allá de EOF
   nbytes := inodo.tamEnBytesLog - offset;
   // leemos sólo los bytes que podemos desde el offset hasta EOF
fsi
```

Contemplaremos los mismos casos que en *mi_write_f()*.
Hay que ir construyendo *buf_original* utilizando primeramente *bread()* para leer un bloque del dispositivo y copiando la porción correspondiente con *memcpy()* al *buf_original*.

---

a cualquier offset (byte lógico) puede haber escrituras que se inicien por debajo de ese último byte escrito y que no lo sobrepasen, y entonces no actualizaríamos el EOF anterior (no sobreescribiríamos el tamaño en bytes lógico guardado), y en cambio otras sí y actualizaríamos el EOF.

Tened en cuenta que las llamadas a *traducir_bloque_inodo()* ahora serán con *reservar* **= 0** y que pueden devolver **-1** si no hay un bloque físico, *nbfisico*, asignado a un determinado bloque lógico . En tal caso **NO hay que hacer el *bread()* del bloque físico** ni por tanto hacer un *memcpy*, simplemente hay que saltar ese bloque pero **acumulando en bytes leídos lo que ocupa ese bloque atravesado**, y seguir iterando.

mi_read_f() debería recibir el *buf_original* inicializado con 0s, si bien eso es responsabilidad de la función o programa que la llame.

Hay que actualizar el *atime*.

Y hay que devolver la cantidad de bytes leídos.

Será en próximos niveles, desde el programa **directorios.c**, cuando usaremos estas funciones para hacer las operaciones de lectura/escritura sobre ficheros/directorios utilizando nombres.

Para probar las funciones anteriores crearemos dos programas ficticios, **escribir.c** y **leer.c** que nos permitan escribir y leer desde consola utilizando el número de inodo como identificador de un fichero concreto (ver Anexo).

3) int mi_stat_f(unsigned int *ninodo, struct STAT *p_stat);*

Devuelve la metainformación de un fichero/directorio (correspondiente al nº de inodo pasado como argumento): tipo, permisos, cantidad de enlaces de entradas en directorio, tamaño en bytes lógicos, *timestamps* y cantidad de bloques ocupados en la zona de datos, es decir todos los campos **menos los punteros**.

Se recomienda definir un tipo estructurado denominado *struct STAT* (podemos considerar que el *struct STAT* es igual que el *struct INODO* pero sin los punteros, no requiere de *padding*).

Para acceder a los campos de la estructura en esta función, al ser pasada por referencia, se usa el operador "→" en vez del ".".

4) int mi_chmod_f(unsigned int *ninodo*, unsigned char *permisos*);

Cambia los permisos de un fichero/directorio (correspondiente al nº de inodo pasado como argumento, *ninodo*) con el valor que indique el argumento *permisos*.

Hay que actualizar *ctime*!

Habrá que hacer un programita ficticio, **permitir.c,** para poder disponer actualmente desde consola de un comando que llame a esta función (ver Anexo) hasta que dispongamos del comando final **mi_chmod** que llame a la función *mi_chmod()* de la capa de directorios, que a su vez llamará a *mi_chmod_f()* de la capa de ficheros.

Anexo

escribir.c, leer.c y permitir.c

Son programas externos **ficticios**, sólo para probar, temporalmente, las funcionalidades de lectura/escritura y cambio de permisos, que involucran funciones de las 3 capas inferiores de nuestra biblioteca del sistema de ficheros, pero estos programas **NO forman parte del sistema de ficheros**.

Estos programas deben comprobar si el número de argumentos es correcto y en caso contrario mostrar la sintaxis.

Han de montar y desmontar el dispositivo virtual.

escribir.c

Escribirá texto en **uno o varios inodos** haciendo uso de *reservar_inodo ('f', 6)* para obtener un nº de inodo, *ninodo*, que mostraremos por pantalla y además utilizaremos como parámetro para *mi_write_f()*.

- Ejemplos de **offsets** para utilizar los diferentes tipos de punteros: 9.000 (⊂BL 8), 209.000 (⊂BL 204), 30.725.000 (⊂BL 30.004), 409.605.000 (⊂BL 400.004) y 480.000.000 (⊂BL 468.750)[2].
- Para indicar el **texto** a escribir tenéis varias opciones a escoger:
  - Pasarlo como argumento escribiéndolo en consola y utilizar la función *strlen()* para calcular su longitud.
  - Pasarlo como argumento haciendo el *cat* de cualquier fichero, por ejemplo un fichero.c de vuestra práctica de la siguiente manera[3]:

"$(cat dir_practica/fichero.c)"

[2] Podéis guardarlos en un array de offsets en el programa
[3] Método aconsejado

- Asignarlo a un *buffer* desde código de esta manera:

  char buffer[tamanyo];

  strcpy (buffer, "blablabla...");

- Pasar como argumento el **nombre de un fichero externo** que contenga el texto[4]

- Un ejemplo de sintaxis para esta función podría ser:

  escribir <nombre_dispositivo> <"$(cat fichero)"> <diferentes_inodos>

  Offsets: 9000, 209000, 30725000, 409605000, 480000000

  Si *diferentes_inodos* = 0 se reserva un solo inodo para todos los offsets.

- Tras escribir en un inodo mostrar el **tamaño en bytes lógico del inodo** y el nº **de bloques ocupados** (podéis obtener los datos llamando a la función *mi_stat_f()*).
- A modo de test, justo después de la llamada a *mi_write_f()* podéis hacer una llamada a *mi_read_f()* con los mismos parámetros para comprobar el funcionamiento de ambas funciones (previa limpieza del *buffer* con un *memset* de 0s. Utilizad el *write(1,...)* del sistema para mostrar el contenido por pantalla (salida estándar: 1). Una vez testeado, **se ha de eliminar esta llamada**.

leer.c

Sintaxis: **leer <nombre_dispositivo> <ninodo>**

Le pasaremos por línea de comandos un nº de inodo, *ninodo*, (además del nombre del dispositivo). Su funcionamiento tiene que ser similar al comando **cat** de Linux, explorando **TODO** el fichero

- La lectura del inodo no se puede hacer de todo el fichero de golpe con *mi_read_f()* ya que nuestro sistema permite ficheros de hasta 16GB y eso no cabría en un *buffer* de memoria, podemos hacerla bloque a bloque hasta llegar al final del fichero (desde *offset* = 0 y avanzando un bloque cada vez). Aunque también podríamos hacer llamadas a *mi_read_f()* con *nbytes* igual al tamaño de varios bloques para mejorar la tasa de transferencia, o con *nbytes* igual a cualquier nº razonable para el buffer (y no necesariamente múltiplo del tamaño de bloque, por ejemplo 1500). **Se ha de guardar el tamaño del buffer de lectura en una variable, *tambuffer*, o constante simbólica para que sea fácilmente modificable.**
- La condición de salida del bucle de lectura es que *mi_read_f()* devuelva 0 bytes leídos:

[4] Si el fichero externo ha sido generado con un editor entonces contendrá un carácter adicional

```
leidos=mi_read_f(ninodo, buffer_texto, offset, tambuffer);
while (leidos > 0){
  ...
  leidos=mi_read_f(ninodo, buffer_texto, offset, tambuffer);
}
```

- El nº total de bytes leídos acumulado en las sucesivas llamadas a *mi_read_f()* tiene que ser igual al nº de bytes lógicos del fichero, *tamEnBytesLog*.
- Para mostrar los resultados por pantalla del contenido se puede utilizar el *write()* de sistema para la salida estándard, **write(1,...)** .

write(1, buffer_texto, leidos);[5]

- Justo antes de cada llamada a *mi_read_f()* **hay que limpiar el *buffer* de lectura con 0s** mediante *memset()*, así la consola os filtrará la basura cuando lo mostréis por pantalla.
- Si redireccionamos la salida estándard de **leer.c** a un fichero desde la línea de comandos (mediante el símbolo ">"), **el tamaño de ese fichero externo** también ha de coincidir con **el tamaño en bytes lógicos del fichero de nuestro sistema**, *tamEnBytesLog*. [6]
- Hay que mostrar el valor del nº de bytes leídos y del tamaño en bytes lógico del inodo. Para ello mostrar ese valor al acabar la lectura utilizando **fprintf(stderr**, ...) o **write(2,...)**, de esta manera no sumará lo que ocupe esa información al fichero externo si hemos redirecionado la lectura del inodo. Ejemplo con *write()*:

```
char string[128];
sprintf(string, "bytes leídos %d\n", leidos);
write(2, string, strlen(string));
```

permitir.c

Sintaxis: **permitir <nombre_dispositivo> <ninodo> <permisos>**

- Validación de sintaxis
- montar dispositivo
- llamada a *mi_chmod_f()* con los argumentos recibidos, convertidos a entero
- desmontar dispositivo

[5] Hay que imprimir la cantidad de bytes realmente leída. Si aquí utilizáis el tamaño del buffer de lectura, puede que en la última iteración no se llene y entonces aunque por pantalla no veáis el resto, sí que irán a parar al fichero externo cuando redireccionemos la salida y ocupará más de lo debido.
[6] Salvo que si el fichero externo ha sido generado con un editor entonces contendrá un carácter adicional por el salto de línea.

Anexo

Punteros inodo	Bloques lógicos	Bytes lógicos
punterosDirectos [0] … [11]	BL 0 … BL 11	0 … 12.287
punterosIndirectos[0]	BL 12 … BL 267	12.288 … 274.431
punterosIndirectos[1]	BL 268 … BL 65.803	273.432 … 67.383.295
punterosIndirectos[2]	BL 65.804 … 16.843.019	67.383.296 … 17.247.252.479

Tabla de rangos para *BLOCKSIZE* = 1024

Tests de prueba

En el aula digital encontraréis los scripts **script1e1.sh**, **script2e1.sh** para descargar y ejecutar (tenéis que darles permisos de ejecución como superusuario). Aquí podréis ver el resultado de su ejecución y compararlo con el vuestro para saber si es correcto.

- **script1e1.sh: leer y escribir texto de menos de 1 bloque en diferentes offsets (9.000, 209.000, 30.725.000, 409.605.000 y 480.000.000) que se encuentran respectivamente en los bloques lógicos 8, 204, 30.004, 400.004 y 468.750**

```
$ ./script1e1.sh
##############################################################################
$ rm disco
$ ./mi_mkfs disco 100000
#inicializamos el sistema de ficheros con 100.000 bloques
##############################################################################
$ ./leer_sf disco
#mostramos solo el SB

DATOS DEL SUPERBLOQUE
posPrimerBloqueMB = 1
posUltimoBloqueMB = 13
posPrimerBloqueAI = 14
posUltimoBloqueAI = 3138
posPrimerBloqueDatos = 3139
posUltimoBloqueDatos = 99999
posInodoRaiz = 0
posPrimerInodoLibre = 1
cantBloquesLibres = 96861
cantInodosLibres = 24999
totBloques = 100000
```

totInodos = 25000

##
$ ./escribir
#consultamos sintaxis comando
Sintaxis: escribir <nombre_dispositivo> <"$(cat fichero)"> <diferentes_inodos>
Offsets: 9000, 209000, 30725000, 409605000, 480000000
Si diferentes_inodos=0 se reserva un solo inodo para todos los offsets
##
$ ./escribir disco 123456789 0
#escribimos el texto "123456789" en los offsets 9000, 209000, 30725000,
#409605000 y 480000000 de un mismo inodo
longitud texto: 9

Nº inodo reservado: 1
offset: 9000
[traducir_bloque_inodo()→ inodo.punterosDirectos[8] = 3139 (reservado BF 3139 para BL 8)]
Bytes escritos: 9
stat.tamEnBytesLog=9009
stat.numBloquesOcupados=1

Nº inodo reservado: 1
offset: 209000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[0] = 3140 (reservado BF 3140 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [192] = 3141 (reservado BF 3141 para BL 204)]
Bytes escritos: 9
stat.tamEnBytesLog=209009
stat.numBloquesOcupados=3

Nº inodo reservado: 1
offset: 30725000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[1] = 3142 (reservado BF 3142 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [116] = 3143 (reservado BF 3143 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [40] = 3144 (reservado BF 3144 para BL 30004)]
Bytes escritos: 9
stat.tamEnBytesLog=30725009
stat.numBloquesOcupados=6

Nº inodo reservado: 1
offset: 409605000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3145 (reservado BF 3145 para punteros_nivel3)]
[traducir_bloque_inodo()→ punteros_nivel3 [5] = 3146 (reservado BF 3146 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [25] = 3147 (reservado BF 3147 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [120] = 3148 (reservado BF 3148 para BL 400004)]
Bytes escritos: 9
stat.tamEnBytesLog=409605009
stat.numBloquesOcupados=10

Nº inodo reservado: 1
offset: 480000000
[traducir_bloque_inodo()→ punteros_nivel3 [6] = 3149 (reservado BF 3149 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [38] = 3150 (reservado BF 3150 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [2] = 3151 (reservado BF 3151 para BL 468750)]

Bytes escritos: 9
stat.tamEnBytesLog=480000009
stat.numBloquesOcupados=13
##
$ ./leer disco 1 > ext1.txt
#leemos el contenido del inodo 1 y lo direccionamos al fichero externo ext1.txt

total_leidos 480000009
tamEnBytesLog 480000009
##
$ ls -l ext1.txt
#comprobamos cuánto ocupa el fichero externo
#(ha de coincidir con el tamaño en bytes lógico del inodo y con los bytes leídos)
-rw-rw-r-- 1 uib uib 480000009 de març 19 14:52 ext1.txt
##
$ ./escribir disco 123456789 1
#escribimos el texto "123456789" en los offsets 9000, 209000, 30725000,
#409605000 y 480000000, de inodos diferentes
longitud texto: 9

Nº inodo reservado: 2
offset: 9000
[traducir_bloque_inodo()→ inodo.punterosDirectos[8] = 3152 (reservado BF 3152 para BL 8)]
Bytes escritos: 9
stat.tamEnBytesLog=9009
stat.numBloquesOcupados=1

Nº inodo reservado: 3
offset: 209000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[0] = 3153 (reservado BF 3153 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [192] = 3154 (reservado BF 3154 para BL 204)]
Bytes escritos: 9
stat.tamEnBytesLog=209009
stat.numBloquesOcupados=2

Nº inodo reservado: 4
offset: 30725000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[1] = 3155 (reservado BF 3155 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [116] = 3156 (reservado BF 3156 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [40] = 3157 (reservado BF 3157 para BL 30004)]
Bytes escritos: 9
stat.tamEnBytesLog=30725009
stat.numBloquesOcupados=3

Nº inodo reservado: 5
offset: 409605000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3158 (reservado BF 3158 para punteros_nivel3)]
[traducir_bloque_inodo()→ punteros_nivel3 [5] = 3159 (reservado BF 3159 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [25] = 3160 (reservado BF 3160 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [120] = 3161 (reservado BF 3161 para BL 400004)]
Bytes escritos: 9
stat.tamEnBytesLog=409605009
stat.numBloquesOcupados=4

Nº inodo reservado: 6
offset: 480000000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3162 (reservado BF 3162 para punteros_nivel3)]
[traducir_bloque_inodo()→ punteros_nivel3 [6] = 3163 (reservado BF 3163 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [38] = 3164 (reservado BF 3164 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [2] = 3165 (reservado BF 3165 para BL 468750)]
Bytes escritos: 9
stat.tamEnBytesLog=480000009
stat.numBloquesOcupados=4
##
$ ./leer disco 2 > ext2.txt
#leemos el contenido del inodo 2 (escrito en el offset 9000) y lo direccionamos
#al fichero externo ext2.txt

total_leidos 9009
tamEnBytesLog 9009
##
$ ls -l ext2.txt
#comprobamos cuánto ocupa el fichero externo ext2.txt
#(ha de coincidir con el tamaño en bytes lógico del inodo 2 y con total_leidos)
-rw-rw-r-- 1 uib uib 9009 de març 19 14:52 ext2.txt
##
$ cat ext2.txt
#usamos el comando cat del sistema para leer el contenido del fichero externo
123456789
##
$ ./leer disco 2
#leemos el contenido de nuestro inodo 2
#(ha de contener lo mismo que el fichero externo ext2.txt)
123456789

total_leidos 9009
tamEnBytesLog 9009
##
$ ./leer disco 5 > ext3.txt
#leemos todo el contenido del inodo 5 (escrito en el offset 409605000) y lo
#direccionamos al fichero externo ext3.txt

```
total_leidos 409605009
tamEnBytesLog 409605009
########################################################################
$ ls -l ext3.txt
#comprobamos cuánto ocupa el fichero externo ext3.txt
#(ha de coincidir con el tamaño en bytes lógico del inodo 5 y con total_leidos)
-rw-rw-r-- 1 uib uib 409605009 de març  19 14:52 ext3.txt
########################################################################
$ cat ext3.txt
#usamos el comando cat del sistema para leer el contenido del fichero externo
123456789
########################################################################
$ ./leer disco 5
#leemos el contenido de nuestro inodo 5
#(ha de contener lo mismo que el fichero externo ext3.txt)
123456789

total_leidos 409605009
tamEnBytesLog 409605009
```

- **script2e1.sh: leer y escribir texto2.txt de más de 1 bloque en diferentes offsets (contenido en el fichero externo texto2.txt) y permisos.** Fijarse que dependiendo del offset, el texto de 3751 bytes ocupará 4 o 5 bloques

```
$ ./script2e1.sh
########################################################################
$ rm disco
$ ./mi_mkfs disco 100000
########################################################################
$ ./escribir disco[7]  ¿Qué es Lorem Ipsum? Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum ha sido el texto de relleno estándar de las industrias
...
Lorem Ipsum que parezca razonable. Este Lorem Ipsum generado siempre estará libre de repeticiones, humor agregado o palabras no características del lenguaje, etc. 1
#escribimos el texto contenido en texto2.txt en los offsets  9000, 209000, 30725000,
#409605000 y 480000000 de inodos diferentes
longitud texto: 3751

Nº inodo reservado: 1
offset: 9000
[traducir_bloque_inodo()→ inodo.punterosDirectos[8] = 3139 (reservado BF 3139 para BL 8)]
[traducir_bloque_inodo()→ inodo.punterosDirectos[9] = 3140 (reservado BF 3140 para BL 9)]
[traducir_bloque_inodo()→ inodo.punterosDirectos[10] = 3141 (reservado BF 3141 para BL 10)]
[traducir_bloque_inodo()→ inodo.punterosDirectos[11] = 3142 (reservado BF 3142 para BL 11)]
```

[7] La orden contiene **"$(cat texto2.txt)"** y nos aparecerá TODO el texto del fichero **texto2.txt** formando parte de la línea de comandos. Aquí sólo se muestra una parte para no ocupar tantas páginas.

```
[traducir_bloque_inodo()→ inodo.punterosIndirectos[0] = 3143 (reservado BF 3143 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [0] = 3144 (reservado BF 3144 para BL 12)]

Bytes escritos: 3751
stat.tamEnBytesLog=12751
stat.numBloquesOcupados=6

Nº inodo reservado: 2
offset: 209000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[0] = 3145 (reservado BF 3145 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [192] = 3146 (reservado BF 3146 para BL 204)]
[traducir_bloque_inodo()→ punteros_nivel1 [193] = 3147 (reservado BF 3147 para BL 205)]
[traducir_bloque_inodo()→ punteros_nivel1 [194] = 3148 (reservado BF 3148 para BL 206)]
[traducir_bloque_inodo()→ punteros_nivel1 [195] = 3149 (reservado BF 3149 para BL 207)]
Bytes escritos: 3751
stat.tamEnBytesLog=212751
stat.numBloquesOcupados=5

Nº inodo reservado: 3
offset: 30725000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[1] = 3150 (reservado BF 3150 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [116] = 3151 (reservado BF 3151 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [40] = 3152 (reservado BF 3152 para BL 30004)]
[traducir_bloque_inodo()→ punteros_nivel1 [41] = 3153 (reservado BF 3153 para BL 30005)]
[traducir_bloque_inodo()→ punteros_nivel1 [42] = 3154 (reservado BF 3154 para BL 30006)]
[traducir_bloque_inodo()→ punteros_nivel1 [43] = 3155 (reservado BF 3155 para BL 30007)]
[traducir_bloque_inodo()→ punteros_nivel1 [44] = 3156 (reservado BF 3156 para BL 30008)]
Bytes escritos: 3751
stat.tamEnBytesLog=30728751
stat.numBloquesOcupados=7

Nº inodo reservado: 4
offset: 409605000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3157 (reservado BF 3157 para punteros_nivel3)]
[traducir_bloque_inodo()→ punteros_nivel3 [5] = 3158 (reservado BF 3158 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [25] = 3159 (reservado BF 3159 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [120] = 3160 (reservado BF 3160 para BL 400004)]
[traducir_bloque_inodo()→ punteros_nivel1 [121] = 3161 (reservado BF 3161 para BL 400005)]
[traducir_bloque_inodo()→ punteros_nivel1 [122] = 3162 (reservado BF 3162 para BL 400006)]
[traducir_bloque_inodo()→ punteros_nivel1 [123] = 3163 (reservado BF 3163 para BL 400007)]
[traducir_bloque_inodo()→ punteros_nivel1 [124] = 3164 (reservado BF 3164 para BL 400008)]
Bytes escritos: 3751
stat.tamEnBytesLog=409608751
stat.numBloquesOcupados=8

Nº inodo reservado: 5
offset: 480000000
```

[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3165 (reservado BF 3165 para punteros_nivel3)]
[traducir_bloque_inodo()→ punteros_nivel3 [6] = 3166 (reservado BF 3166 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [38] = 3167 (reservado BF 3167 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [2] = 3168 (reservado BF 3168 para BL 468750)]
[traducir_bloque_inodo()→ punteros_nivel1 [3] = 3169 (reservado BF 3169 para BL 468751)]
[traducir_bloque_inodo()→ punteros_nivel1 [4] = 3170 (reservado BF 3170 para BL 468752)]
[traducir_bloque_inodo()→ punteros_nivel1 [5] = 3171 (reservado BF 3171 para BL 468753)]
Bytes escritos: 3751
stat.tamEnBytesLog=480003751
stat.numBloquesOcupados=7
##
$ ./leer disco 2 > ext4.txt
#leemos el contenido del inodo 2 (escrito en el offset 209000) y lo direccionamos
#al fichero externo ext4.txt

total_leidos 212751
tamEnBytesLog 212751
##
$ ls -l ext4.txt
#comprobamos cuánto ocupa el fichero externo ext4.txt
#(ha de coincidir con el tamaño en bytes lógico del inodo 2 y con total_leidos)
-rw-rw-r-- 1 uib uib 212751 de març 19 14:56 ext4.txt
##
$ cat ext4.txt
#usamos el cat del sistema para leer el contenido de nuestro fichero direccionado
#No hay que mostrar basura

¿Qué es Lorem Ipsum?

Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum ha sido el texto de relleno estándar de las industrias desde el año 1500, cuando un impresor (N. del T. persona que se dedica a la imprenta) desconocido usó una galería de textos y los mezcló de tal manera que logró hacer un libro de textos especimen. No sólo sobrevivió 500 años, sino que tambien ingresó como texto de relleno en documentos electrónicos, quedando esencialmente igual al original. Fue popularizado en los 60s con la creación de las hojas "Letraset", las cuales contenian pasajes de Lorem Ipsum, y más recientemente con software de autoedición, como por ejemplo Aldus PageMaker, el cual incluye versiones de Lorem Ipsum.
¿Por qué lo usamos?

Es un hecho establecido hace demasiado tiempo que un lector se distraerá con el contenido del texto de un sitio mientras que mira su diseño. El punto de usar Lorem Ipsum es que tiene una distribución más o menos normal de las letras, al contrario de usar textos como por ejemplo "Contenido aquí, contenido aquí". Estos textos hacen parecerlo un español que se puede leer. Muchos paquetes de autoedición y editores de páginas web usan el Lorem Ipsum como su texto por defecto, y al hacer una búsqueda de "Lorem Ipsum" va a dar por resultado muchos sitios web que usan este texto si se encuentran en estado de desarrollo. Muchas versiones han evolucionado a través de los años, algunas veces por accidente, otras veces a propósito (por ejemplo insertándole humor y cosas por el estilo).

¿De dónde viene?

Al contrario del pensamiento popular, el texto de Lorem Ipsum no es simplemente texto aleatorio. Tiene sus raices en una pieza cl´sica de la literatura del Latin, que data del año 45 antes de Cristo, haciendo que este adquiera mas de 2000 años de antiguedad. Richard McClintock, un profesor de Latin de la Universidad de Hampden-Sydney en Virginia, encontró una de las palabras más oscuras de la lengua del latín, "consecteur", en un pasaje de Lorem Ipsum, y al seguir leyendo distintos textos del latín, descubrió la fuente indudable. Lorem Ipsum viene de las secciones 1.10.32 y 1.10.33 de "de Finnibus Bonorum et Malorum" (Los Extremos del Bien y El Mal) por Cicero, escrito en el año 45 antes de Cristo. Este libro es un tratado de teoría de éticas, muy popular durante el Renacimiento. La primera linea del Lorem Ipsum, "Lorem ipsum dolor sit amet..", viene de una linea en la sección 1.10.32

El trozo de texto estándar de Lorem Ipsum usado desde el año 1500 es reproducido debajo para aquellos interesados. Las secciones 1.10.32 y 1.10.33 de "de Finibus Bonorum et Malorum" por Cicero son también reproducidas en su forma original exacta, acompañadas por versiones en Inglés de la traducción realizada en 1914 por H. Rackham.
¿Dónde puedo conseguirlo?

Hay muchas variaciones de los pasajes de Lorem Ipsum disponibles, pero la mayoría sufrió alteraciones en alguna manera, ya sea porque se le agregó humor, o palabras aleatorias que no parecen ni un poco creíbles. Si vas a utilizar un pasaje de Lorem Ipsum, necesitás estar seguro de que no hay nada avergonzante escondido en el medio del texto. Todos los generadores de Lorem Ipsum que se encuentran en Internet tienden a repetir trozos predefinidos cuando sea necesario, haciendo a este el único generador verdadero (válido) en la Internet. Usa un diccionario de mas de 200 palabras provenientes del latín, combinadas con estructuras muy útiles de sentencias, para generar texto de Lorem Ipsum que parezca razonable. Este Lorem Ipsum generado siempre estará libre de repeticiones, humor agregado o palabras no características del lenguaje, etc.**##**
$ ./permitir
#mostramos sintaxis de permitir
Sintaxis: permitir <nombre_dispositivo> <ninodo> <permisos>
##
$ ./permitir disco 2 0
#cambiamos permisos del inodo 2 a 0
##
$ ./leer disco 2
#intentamos leer inodo 2 con permisos=0
No hay permisos de lectura

total_leidos 0
tamEnBytesLog 212751