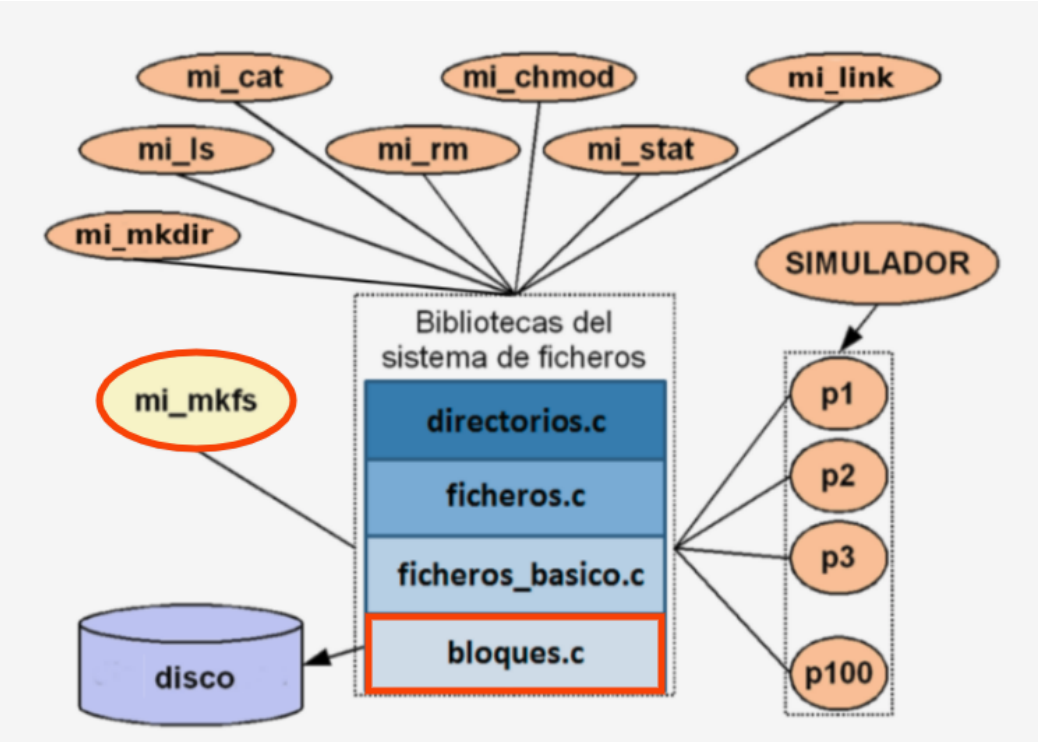
Nivel 1: bloques.c {bmount(), bumount(), bwrite(), bread()} y mi_mkfs.c

Un **fichero lógico** se compone de una secuencia de **bloques lógicos**.

El **tamaño del bloque lógico** se corresponde con el **tamaño del bloque físico**, y es el mismo para todo el sistema de ficheros (BLOCKSIZE) y suele ser múltiplo de 512 bytes[1] . En nuestro dispositivo virtual los bloques serán de 1024 bytes.

Vamos a programar en **bloques.c** las funciones básicas para E/S de bloques y luego **mi_mkfs.c** será el programa que **formateará** nuestro sistema de ficheros y hará uso de tales funciones para montar/desmontar el dispositivo virtual y **leer/escribir por bloques**.



1) int bmount(const char *camino);

Llamar a la función open() del sistema para obtener el descriptor del fichero que se usará como dispositivo virtual.

[1] El tamaño elegido para el bloque va a influir en las prestaciones globales del sistema.

- Bloques grandes ⇒ velocidad de transferencia entre el disco y la memoria grande.
- Si demasiado grandes ⇒ capacidad de almacenamiento del disco desaprovechada cuando abundan los archivos pequeños que no llegan a ocupar un bloque completo.

int open(const char *camino, int oflags, mode_t mode);

El nombre del fichero va en el argumento de la función (nos lo proporcionará el administrador del sistema desde consola cuando inicialice el sistema de ficheros a través de mi_mkfs.c).

Valores para oflags:

O_RDONLY	Abre el fichero sólo para lectura
O_WRONLY	Abre el fichero sólo para escritura
O_RDWR	Abre el fichero para lectura y escritura
O_APPEND	Añade información al final del fichero
O_TRUNC	Inicialmente borra todos los datos del fichero
O_CREAT	Si el fichero no existe lo crea. En este caso se requiere el 3er parámetro (permisos).
O_EXCL	Combinado con la opción O_CREAT, asegura que el que lo llama debe crear el fichero, si ya existe la llamada fallará.

Podemos abrirlo como **O_RDWR|O_CREAT**.

Los permisos se representan en octal, por ej: **0666** significa que damos permiso de lectura (r) y escritura (w) a usuario, grupo y otros: rw-rw-rw-. [+Información sobre permisos].

octal	binario	permiso
7	111	rwx
6	110	rw-
5	101	r-x
4	100	r--
3	011	-wx
2	010	-w-
1	001	--x
0	000	---

Un proceso o programa tiene asignada una tabla de descriptores de ficheros (nº entero de 0-19, 0: entrada estándar, 1: salida estándar, 2: salida de error estándar). La función *open()* retornará el **descriptor del fichero** (el más bajo libre en la tabla de descriptores) para ser usado en las siguientes operaciones de E/S.

Abrir un fichero es una llamada al sistema y puede producir errores que hay que gestionar. [+Información sobre llamadas al sistema.]

```
if (descriptor == -1) {
        ...  //error
}
```

bmount() devuelve -1 (o EXIT_FAILURE) si ha habido error o el descriptor si ha ido bien.

2) int bumount();

Básicamente llama a la función close() para liberar el descriptor de ficheros.

```
int close(int descriptor);
```

La función *bumount()* devuelve 0 (o EXIT_SUCCESS) si se ha cerrado el fichero correctamente y -1 (o EXIT_FAILURE) en caso contrario.

3) int bwrite(unsigned int nbloque, const void *buf);

Escribe el contenido de un buffer de memoria apuntado por *buf (que tendrá el tamaño de un bloque) en el bloque del dispositivo virtual especificado por el argumento nbloque.

Hay que utilizar la función write() precedida de la función lseek() (el desplazamiento será el nº de bloque * tamaño del bloque, y se comenzará a contar, como punto de referencia, desde el inicio del fichero: SEEK_SET).

```
off_t lseek(int descriptor, off_t desplazamiento, int punto_de_referencia);

size_t write(int descriptor, const void *buf, size_t nbytes);
```

La función bwrite() devuelve el nº de bytes que ha podido escribir, o -1 (o EXIT_FAILURE) si se produce un error.

4) int bread(unsigned int nbloque, void *buf);

Lee del dispositivo virtual el bloque especificado por nbloque. Copia su contenido en un buffer de memoria apuntado por *buf.

Hay que utilizar la función read() precedida de la función lseek().

size_t read(int descriptor, void *buf, size_t nbytes);

El puntero del fichero quedará indicando el siguiente byte a leer.

La función *bread()* devuelve el nº de bytes leídos o, -1 (o EXIT_FAILURE) si se produce un error.

OBSERVACIONES:

- Es conveniente que el descriptor del fichero se almacene como variable global estática (para que sólo pueda ser accedida en **bloques.c**).

static int descriptor = 0;

- El tamaño de bloque, BLOCKSIZE, ha de ser una constante, en nuestro caso 1024.
- En bloques.h hay que incluir las cabeceras básicas fundamentales, y la declaración de funciones y constantes

```
// bloques.h

#include <stdio.h>  //printf(), fprintf(), stderr, stdout, stdin
#include <fcntl.h> //O_WRONLY, O_CREAT, O_TRUNC
#include <sys/stat.h> //S_IRUSR, S_IWUSR
#include <stdlib.h>  //exit(), EXIT_SUCCESS, EXIT_FAILURE, atoi()
#include <unistd.h> // SEEK_SET, read(), write(), open(), close(), lseek()
#include <errno.h>  //errno
#include <string.h> // strerror()

#define BLOCKSIZE 1024 // bytes

int bmount(const char *camino);
int bumount();
int bwrite(unsigned int nbloque, const void *buf);
int bread(unsigned int nbloque, void *buf);
```

- En **bloques.c** y en **mi_mkfs.c** tenéis que incluir luego esta cabecera:

```
#include "bloques.h"
```

Vamos a escribir una primera versión del programa **mi_mkfs.c** que pruebe las funciones de **bloques.c**:

- El programa **mi_mkfs.c** sirve para crear el dispositivo virtual con el tamaño adecuado. Debe ser llamado desde la línea de comandos con los siguientes parámetros para dar nombre al dispositivo virtual y determinar la cantidad de bloques de que dispondrá nuestro sistema de ficheros:

  $ ./mi_mkfs <nombre_dispositivo> <nbloques>

  Recordemos que los parámetros se recuperan con

  ```
  int main(int argc, char **argv)
  ```

  donde:

  argc: número de parámetros

  argc=3

  argv: vector de punteros a los parámetros:

  - argv[0]="mi_mkfs"
  - argv[1]=nombre_dispositivo
  - argv[2]=nbloques (puede sernos útil la función *atoi()* para obtener el valor numérico a partir del string)

- El primer paso es montar (*bmount()*) el fichero que se usará como dispositivo virtual.
- Es necesario llamar a *bwrite()* el número de veces necesario (indicado por el valor de cantidad_bloques, pasado como parámetro) para inicializar el fichero usado como dispositivo virtual.
- En este caso, el buffer de memoria empleado puede ser un array de tipo ***unsigned char*** del tamaño de un bloque (lo inicializaremos a 0 con la función memset(), hay que incluir <string.h> para utilizarla).
- El último paso es desmontar (*bumount()*) el fichero usado como dispositivo virtual.

OBSERVACIONES:

- A cada función hay que asociarle un comentario que explique para qué sirve, qué son cada uno de los parámetros de entrada, y qué devuelve. Opcionalmente también se añadirá un listado de a qué funciones llama y por

cuáles es llamada (eso se irá completando a lo largo del desarrollo de la práctica)

- De momento podéis compilar conjuntamente mi_mkfs.c y bloques.c desde la línea de comandos con gcc de la siguiente manera:

  $ gcc -o mi_mkfs mi_mkfs.c bloques.c

- Y luego ejecutarlo pasándole los parámetros correspondientes

  $ ./mi_mkfs <nombre_dispositivo> <nbloques>

- Pero es recomendable utilizar Make (**será imprescindible a medida que vayamos creando más y más programas**)

```
CC=gcc
CFLAGS=-c -g -Wall -std=gnu99
#LDFLAGS=-pthread

SOURCES=mi_mkfs.c bloques.c #ficheros_basico.c leer_sf.c ficheros.c escribir.c leer.c
truncar.c permitir.c directorios.c mi_mkdir.c mi_chmod.c mi_ls.c mi_link.c mi_escribir.c
mi_cat.c mi_stat.c mi_rm.c semaforo_mutex_posix.c simulacion.c verificacion.c
LIBRARIES=bloques.o #ficheros_basico.o ficheros.o directorios.o
semaforo_mutex_posix.o
INCLUDES=bloques.h #ficheros_basico.h ficheros.h directorios.h
semaforo_mutex_posix.h simulacion.h
PROGRAMS=mi_mkfs #leer_sf escribir leer truncar permitir mi_mkdir mi_chmod mi_ls
mi_link mi_escribir mi_cat mi_stat mi_rm  simulacion verificacion
OBJS=$(SOURCES:.c=.o)

all: $(OBJS) $(PROGRAMS)

$(PROGRAMS): $(LIBRARIES) $(INCLUDES)
  $(CC) $(LDFLAGS) $(LIBRARIES) $@.o -o $@

%.o: %.c $(INCLUDES)
  $(CC) $(CFLAGS) -o $@ -c $<

.PHONY: clean
clean:
  rm -rf *.o *~ $(PROGRAMS) disco* ext*
```

- Para depurar el código, es recomendable utilizar gdb (o alguna de sus variantes gráficas: Nemiver, por ejemplo) o utilizar las posibilidades de depuración que ofrece el editor Visual Studio Code (hay un curso gratuito en Udemy para sacar partido del editor, que incluye el depurador y el manejo de versiones ligado a Git).

- Para analizar el contenido del fichero usado como dispositivo virtual, es recomendable utilizar cualquier editor hexadecimal: Okteta, o GHex, por ejemplo.

TESTS DE PRUEBA

$ ./mi_mkfs disco 100000

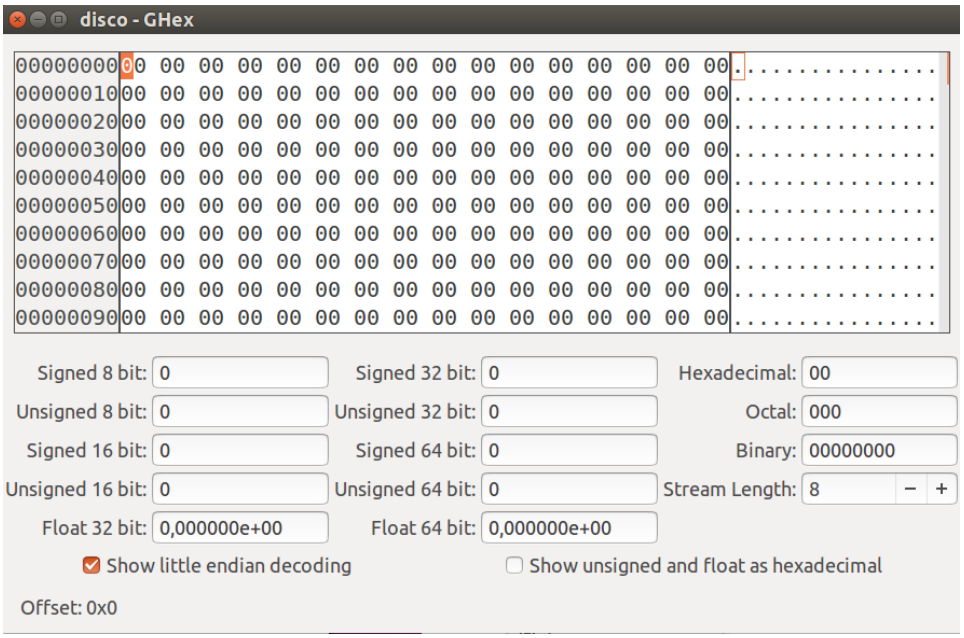
$ ls -l disco #ha de ocupar 102.400.000 bytes

-rw-rw-rw- 1 uib uib 102400000 de febr. 22 11:13 disco

También podéis comprobar el tamaño mirando las propiedades del fichero en el sistema de ventanas [2]:

Y luego abrirlo con GHex u Okteta y comprobar que está todo lleno de 0s:

[2] Aunque es preferible en esta asignatura acostumbrarse a manejar la consola en vez del sistema de ventanas

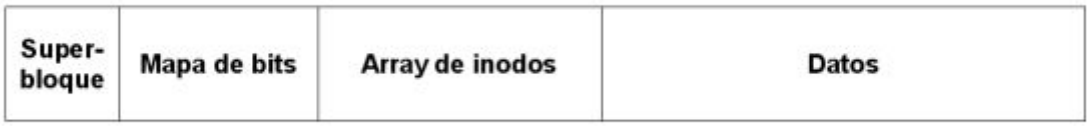


Recursos adicionales:

- Guía rápida de gcc y gdb
- Manual de Nemiver
- C Code Style Guidelines

Nivel 2: ficheros_basico.c {tamMB(), tamAI(), initSB(), initMB(), initAI()}

La estructura interna de nuestro sistema de ficheros será la siguiente:

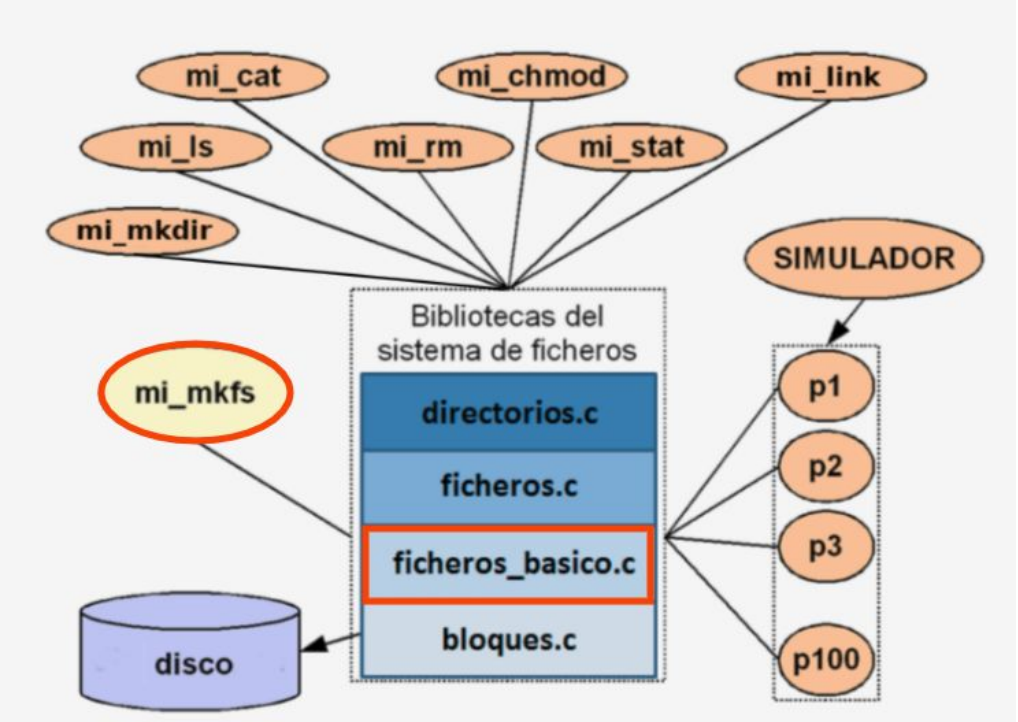


Los 3 primeros componentes constituyen los metadatos.

- El **superbloque** es un bloque que contiene información general sobre el sistema de ficheros. En nuestro sistema su ubicación, posSB, será el bloque 0[1], y su tamaño, tamSB, será de 1 bloque (BLOCKSIZE). Los datos del superbloque se inicializarán con la función initSB().
- El **mapa de bits** nos servirá para gestionar el espacio libre en nuestro sistema de ficheros. Contendrá un bit por cada bloque del sistema de ficheros, que valdrá 0 para los bloques libres y 1 para bloques ocupados. El tamaño del mapa de bits nos lo calculará la función tamMB(). El mapa de bits se inicializará con la función initMB().
- Un **inodo** contiene las características (metadatos) de un directorio o fichero del sistema de ficheros.
  - Los inodos se guardan en el **array de inodos**. La cantidad de inodos por nosotros definida (de forma heurística) indica la máxima cantidad de directorios y ficheros que pueden llegar a existir, en nuestro caso será igual a nbloques/4. El tamaño del array de inodos nos lo calculará la función tamAI().
  - Dentro del array de inodos, los **inodos libres** se organizan como una **lista enlazada**, mientras que los inodos ocupados se corresponden a directorios o ficheros existentes. La función initAI() se encargará de crear inicialmente esa lista enlazada.
- El contenido de los directorios (entradas) y el de los ficheros en sí se almacenan en la zona de **datos**.

Comenzamos a desarrollar en **ficheros_basico.c** las funciones de tratamiento básico del sistema de ficheros y también definiremos la estructura del superbloque y la estructura de un inodo. Después actualizaremos **mi_mkfs.c** para llamar a las funciones de inicialización de cada zona de metadatos, que hemos desarrollado en ficheros_basico.c.

[1] Si tuviéramos un dispositivo real de memoria secundaria, el bloque 0 estaría reservado como bloque de arranque del disco



SUPERBLOQUE

Lo declararemos como un *struct* con los siguientes campos, de tipo *unsigned int*:

```
struct superbloque {
   unsigned int posPrimerBloqueMB;          // Posición del primer bloque del mapa de bits en el SF
   unsigned int posUltimoBloqueMB;          // Posición del último bloque del mapa de bits en el SF
   unsigned int posPrimerBloqueAI;          // Posición del primer bloque del array de inodos en el SF
   unsigned int posUltimoBloqueAI;          // Posición del último bloque del array de inodos en el SF
   unsigned int posPrimerBloqueDatos;       // Posición del primer bloque de datos en el SF
   unsigned int posUltimoBloqueDatos;       // Posición del último bloque de datos en el SF
   unsigned int posInodoRaiz;               // Posición del inodo del directorio raíz en el AI [2]
   unsigned int posPrimerInodoLibre;        // Posición del primer inodo libre en el AI
   unsigned int cantBloquesLibres;          // Cantidad de bloques libres del SF
   unsigned int cantInodosLibres;           // Cantidad de inodos libres del SF
   unsigned int totBloques;                 // Cantidad total de bloques del SF
   unsigned int totInodos;                  // Cantidad total de inodos del SF
   char padding[BLOCKSIZE - 12 * sizeof(unsigned int)]; // Relleno para que ocupe 1 bloque
};
```

Las variables que declaremos de tipo struct superbloque serán locales en cada función y se tendrá que acceder al disco para obtener el valor de los campos que varíen a lo largo

[2] Es la posición realtiva dentro del array de inodos, no es el nº de bloque del sistema de ficheros

de la vida del sistema, ya que cuando lancemos varios procesos, si fuese global, cada proceso tendría su propia copia del SB y se trata de datos comunes a todo el sistema. Una mejora del sistema sería guardar el SB en memoria compartida usando mmap().

Para los inodos[3] también emplearemos un *struct* (declarado en ficheros_basico.h), inodo, con los siguientes campos (no llegan a ocupar los 128 bytes disponibles, por lo que habrá espacio de sobra para añadir otros campos que nos pudieran interesar durante la práctica):

```
struct inodo {     // comprobar que ocupa 128 bytes haciendo un sizeof(inodo)!!!
  char tipo;     // Tipo ('l':libre, 'd':directorio o 'f':fichero)
  char permisos; // Permisos (lectura y/o escritura y/o ejecución)
  /* Por cuestiones internas de alineación de estructuras, si se está utilizando
   un tamaño de palabra de 4 bytes (microprocesadores de 32 bits):
  unsigned char reservado_alineacion1 [2];
  en caso de que la palabra utilizada sea del tamaño de 8 bytes
  (microprocesadores de 64 bits): unsigned char reservado_alineacion1 [6]; */
  char reservado_alineacion1[6];
  time_t atime; // Fecha y hora del último acceso a datos: atime
  time_t mtime; // Fecha y hora de la última modificación de datos: mtime
  time_t ctime; // Fecha y hora de la última modificación del inodo: ctime

  /* comprobar el tamaño del tipo time_t para vuestra plataforma/compilador:
  printf ("sizeof time_t is: %d\n", sizeof(time_t)); */

  unsigned int nlinks;            // Cantidad de enlaces de entradas en directorio
  unsigned int tamEnBytesLog;      // Tamaño en bytes lógicos. Se actualizará al escribir si crece
  unsigned int numBloquesOcupados; // Cantidad de bloques ocupados zona de datos

  unsigned int punterosDirectos[12];  // 12 punteros a bloques directos
  unsigned int punterosIndirectos[3]; /* 3 punteros a bloques indirectos:
  1 indirecto simple, 1 indirecto doble, 1 indirecto triple */

  /* Utilizar una variable de alineación si es necesario para vuestra plataforma/compilador */
  char
     padding[INODOSIZE - 2 * sizeof(unsigned char) - 3 * sizeof(time_t) - 18 * sizeof(unsigned
int) - 6 * sizeof(unsigned char)];
  // Hay que restar también lo que ocupen las variables de alineación utilizadas!!!
};
```

Una estructura mejor que la del inodo con el padding para que sea exactamente 128 bytes y funcione en todas las arquitecturas es utilizar en realidad una **unión** de la estructura del inodo y un inodo (ver Anexo). Su utilización es voluntaria.

[3] https://www.youtube.com/watch?v=i4VXNX-8BO0&t=282s

Los **permisos** se manejan como los permisos básicos característicos de GNU/Linux: lectura ('r'), escritura ('w') y ejecución ('x'). Sólo tendremos en cuenta los de usuario.

Así que las posibles combinaciones de permisos son las que van de 000 a 111 en binario (de 0 a 7 en octal): el primer bit para la 'r', el segundo bit para la 'w' y el tercer bit para la 'x'.

Para cada ***timestamp*** (fecha y hora) atime, mtime y ctime, se pueden utilizar los tipos y las funciones declaradas en *time.h*. Se pueden inicializar con la función time(NULL).

Hay que estar familiarizado con el concepto de fecha y hora en forma *epoch*: tipo *time_t* [4].

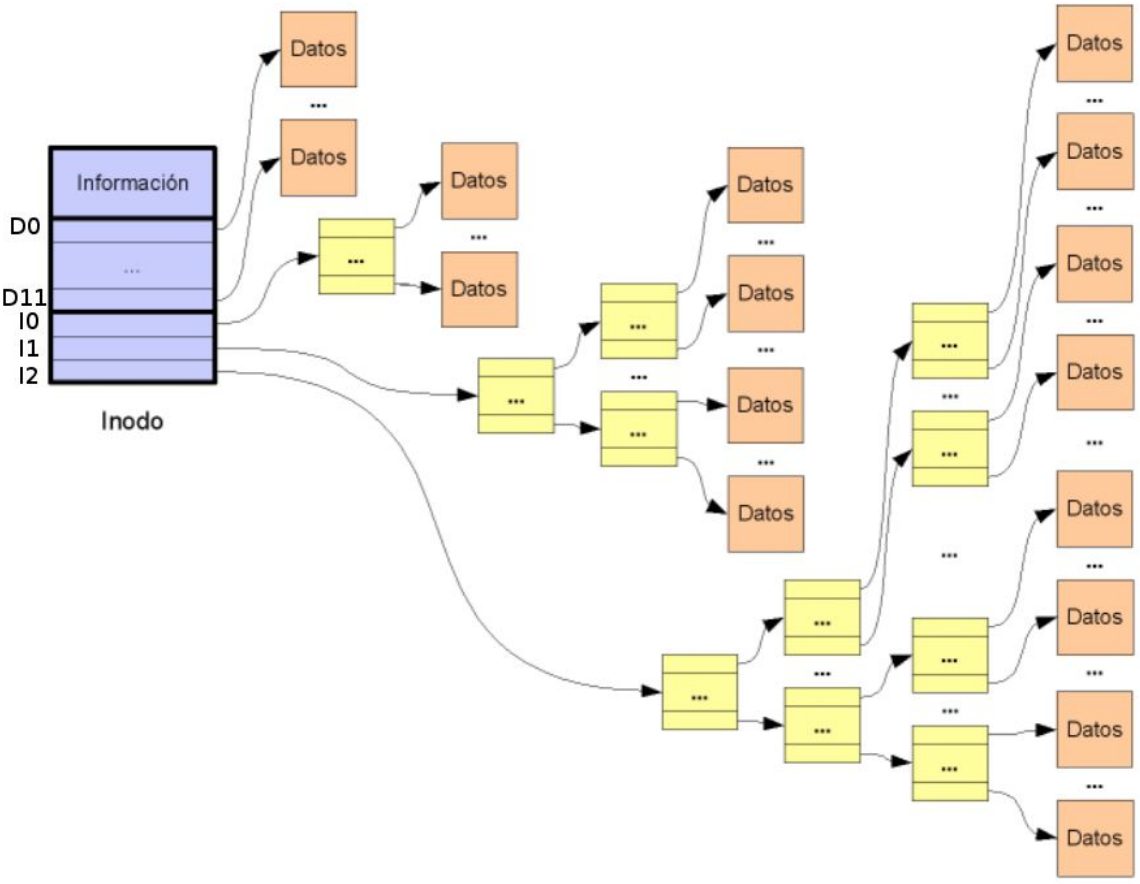
El **nº de enlaces** por defecto es 1. Aumentará con el uso de la función mi_link() y se decrementará con mi_unlink() de la capa de directorios.

El **tamaño en bytes lógico** nos indica la posición del byte lógico más alejado en el fichero, independientemente de cuantos bytes se hayan escrito. Por ejemplo si el byte lógico más alejado está en el offset 10568 (en bytes), entonces el tamEnBytesLog = 10569

La **cantidad de bloques ocupados** incluye tanto los bloques de datos propiamente dichos como los bloques índices de la zona de datos correspondientes a tal inodo.

Los **punteros Directos e Indirectos de varios niveles** nos permiten hacer la correspondencia de los bloques lógicos del inodo con los bloques físicos del dispositivo.

[4] time_t es un long int que contiene un *timestamp* expresado en segundos después del inicio de la época UNIX (1 de Enero de 1970 00:00:00 GMT).



Los **punteros directos D0-D11** contienen la dirección (nº de bloque del dispositivo virtual) de los bloques físicos de datos correspondientes a los bloques lógicos del 0 al 11. De esta manera, si BLOCKSIZE = 1.024, para ficheros pequeños de hasta 12KBs se podría acceder directamente a su contenido, a través de los punteros directos.

El **puntero indirecto I0** contiene la dirección (nº de bloque del dispositivo virtual) de un bloque índice con NPUNTEROS, siendo NPUNTEROS = BLOCKSIZE / sizeof(unsigned int). Para BLOCKSIZE = 1024, NPUNTEROS = 256. Esos punteros son las direcciones (nº de bloque) de los bloques físicos de datos correspondientes a los bloques lógicos del 12 al INDIRECTOS0 - 1, siendo INDIRECTOS0 = 12 + NPUNTEROS. Para BLOCKSIZE = 1.024, nos direccionaría los bloques lógicos del 12 al 267 (12+256-1).

El **puntero indirecto I1** contiene la dirección del bloque físico índice con NPUNTEROS a los bloques índices de NPUNTEROS a los bloques físicos de datos correspondientes a los bloques lógicos del 268 al INDIRECTOS1 - 1, siendo INDIRECTOS1 = INDIRECTOS0 + $NPUNTEROS^2$. Para BLOCKSIZE = 1.024, nos direccionaría los bloques lógicos del 268 al 65.803 ($268+256^2-1$).

El **puntero indirecto I2** contiene la dirección del bloque físico índice con NPUNTEROS a los bloques físicos de índices de NPUNTEROS, los cuales apuntan a los bloques físicos de índices que contienen la dirección de los bloques físicos de datos correspondientes a los bloques lógicos del 65.804 al INDIRECTOS2 - 1, siendo INDIRECTOS2 = INDIRECTOS1 + NPUNTEROS3. Para BLOCKSIZE = 1.024, nos direccionaría los bloques lógicos del 65.804 al 16.843.019 ($65.804 + 256^3 - 1$).

Con esta estructura de punteros del inodo, para un tamaño de bloque de 1KB

podríamos tener ficheros de hasta 16.842.020 KBs (o sea de un tamaño lógico de unos 16 GBs!!!).

Veamos ahora cómo se ha de llevar a cabo la implementaicón de las funciones de este nivel:

1)int tamMB(unsigned int nbloques);

Hay que programar esta función para calcular el tamaño, en bloques, necesario para el mapa de bits.

Dado que cada bit representa un bloque y que los bits se agrupan de 8 en 8 para constituir bytes, y los bytes se agrupan en bloques de tamaño BLOCKSIZE, el tamaño del mapa de bits lo obtendremos mediante:

(nbloques / 8bits) / BLOCKSIZE

Utilizaremos el operador módulo % para saber si necesitamos esa cantidad justa o si necesitamos añadir un bloque adicional para los bytes restantes (el resto de la división).

Si por ejemplo tenemos 100.000 bloques de tamaño 1KB , el tamaño del MB será: (100.000/8)/1.024=12→ 13 bloques (hemos incrementado en 1 el resultado de la división entera con 1024 porque el módulo no es 0)

2)int tamAI(unsigned int ninodos);

Hay que determinar de manera heurística la cantidad de inodos de nuestro sistema (nbloques/2, nbloques/4, nbloques/8...), cantidad que se mantendrá durante toda la vida del sistema de ficheros. En nuestro sistema de ficheros usaremos ninodos=nbloques/4. El programa mi_mkfs.c le pasará este dato a esta función como parámetro al llamarla.

Una vez determinada la cantidad de nodos del sistema, ya podemos calcular el tamaño del array de inodos, en bloques:

(ninodos * INODOSIZE) / BLOCKSIZE

Utilizaremos el operador módulo % con BLOCKSIZE para saber si necesitamos esa cantidad justa o si necesitamos añadir un bloque adicional para el resto, resultado de la división.

El tamaño de nuestros inodos (INODOSIZE) será de **128 bytes**, así que en un bloque de tamaño 1024 bytes nos caben exactaente 8 inodos..

Ejemplos:

- ninodos= 5.000, INODOSIZE= 128 bytes, BLOCKSIZE = 1.024 ⇒ tamAI = 625 bloques
- ninodos= 125.500, INODOSIZE= 128 bytes, BLOCKSIZE = 1.024 ⇒ tamAI = 15.688 bloques
- ninodos= 25.000, INODOSIZE= 128 bytes, BLOCKSIZE = 1.024 ⇒ tamAI = 3.125 bloques

3)int initSB(unsigned int nbloques, unsigned int ninodos);

Basándonos en las funciones *tamMB()* y *tamAI()* vamos a definir una función que permita rellenar los datos básicos del superbloque.

Se trata de definir una variable de tipo superbloque que se vaya rellenando con la información pertinente[5]:

- **Posición[6] del primer bloque del mapa de bits**
  SB.posPrimerBloqueMB = posSB + tamSB //posSB = 0, tamSB = 1
  SB.posPrimerBloqueMB = 0+1 = 1
- **Posición del último bloque del mapa de bits**
  SB.posUltimoBloqueMB = SB.posPrimerBloqueMB+tamMB(nbloques)-1
  tamMB(100000) = (100000/8/1024)+1 = 13
  SB.posUltimoBloqueMB = 1+13-1 = 13
- **Posición del primer bloque del array de inodos**
  SB.posPrimerBloqueAI = SB.posUltimoBloqueMB+1
  SB.posPrimerBloqueAI = 13+1 = 14
- **Posición del último bloque del array de inodos**
  SB.posUltimoBloqueAI = SB.posPrimerBloqueAI+tamAI(ninodos)-1
  tamAI(ninodos) = 25000*128/1.024 = 3125
  SB.posUltimoBloqueAI = 14+3125-1 = 3138
- **Posición del primer bloque de datos**
  SB.posPrimerBloqueDatos = SB.posUltimoBloqueAI+1
  SB.posPrimerBloqueDatos = 3138+1 = 3139
- **Posición del último bloque de datos**
  SB.posUltimoBloqueDatos = nbloques-1
  SB.posUltimoBloqueDatos = 100000-1 = 99999
- **Posición del inodo del directorio raíz en el array de inodos**
  SB.posInodoRaiz = 0
- **Posición del primer inodo libre en el array de inodos**
  - Inicialmente SB.posPrimerInodoLibre = 0.
  - Tras crear el Directorio raíz (Nivel 3) pasará a valer 1.
  - Posteriormente se irá actualizando para apuntar a la cabeza de la lista de inodos libres (mediante las llamadas a las funciones reservar_inodo() y liberar_inodo())
- **Cantidad de bloques libres en el SF**
  - Inicialmente: SB.cantBloquesLibres = nbloques
  - Cuando indiquemos en el mapa de bits los bloques que ocupan los metadatos (el SB, el propio MB y el AI), restaremos esos bloques de la cantidad de bloques libres (Nivel 3)
  - Al reservar un bloque ⇒ SB.cantBloquesLibres--

[5] Debajo de cada asignación genérica os pongo un ejemplo numérico particular para nbloques=100.000, BLOCKSIZE=1024 e INODOSIZE = 128
[6] Todas las posiciones se refieren al nº de bloque

  - Al liberar un bloque ⇒ SB.cantBloquesLibres++
- **Cantidad de inodos libres en el array de inodos**
  - Inicialmente: SB.cantInodosLibres = ninodos (el 1er inodo será para el directorio raíz)
  - Al reservar un inodo ⇒ SB.cantInodosLibres--
  - Al liberar un inodo ⇒ SB.cantInodosLibres++
- **Cantidad total de bloques**
  - Se pasará como argumento en la línea de comandos al inicializar el sistema ($ ./mi_mkfs <nombre_fichero> <nbloques>) y lo recibimos como parámetro

    SB.totBloques = nbloques
- **Cantidad total de inodos**
  - Determinada por el administrador del sistema de forma heurística (ninodos = nbloques/4) y recibida por parámetro

    SB.totinodos = ninodos
  - Se indicará su valor en mi_mkfs.c y se pasará también como parámetro a la función de inicialización del array de inodos

Al finalizar las inicializaciones de los campos, se escribe la estructura en el bloque posSB mediante la función bwrite().

4)int initMB();

En este nivel, de momento, simplemente pondremos a 0 todos los bits del mapa de bits.

Para ello utilizaremos un buffer (será un array de tipo *unsigned char* del tamaño de un bloque) con todos los bits a cero. La función *memset()* puede sernos útil para asignar de golpe un valor a todos los elementos de un array.

El contenido del buffer se escribe en los bloques correspondientes al mapa de bits[7], mediante sucesivas llamadas a la función *bwrite()* (habrá que leer primeramente el superbloque para obtener la localización del mapa de bits).

En el nivel siguiente, cuando ya dispongamos de la función escribir_bit(), la modificaremos para que tenga en cuenta los bloques ocupados por los metadatos, o sea el superbloque, el mapa de bits y el array de inodos (y restaremos esos bloques al total de bloques libres en el superbloque, si no los hemos restado al inicializarlo).

5)int initAI();

Esta función se encargará de inicializar la lista de inodos libres. Dado que al principio todos los inodos están libres, hay que crear una función que enlace todos los inodos entre sí. Cuando el sistema de ficheros esté en funcionamiento, serán las funciones reservar_inodo() y liberar_inodo(), del siguiente nivel, las que gestionarán esta lista, actualizándola siempre por la cabecera.

[7] Sabemos cuántos bloques ocupa el mapa de bits gracias a la función *tamMB()*, y también sabemos cuál es la primera posición del mapa de bits en el dispositivo y cuál es la última (esa información está en el superbloque)

No es necesario definir nuevos campos en el inodo para apuntar al siguiente inodo libre, dado que la mayoría sólo tienen sentido cuando el inodo está ocupado. **Utilizaremos el campo de punterosDirectos[0] para enlazar la lista de inodos libres.**

De nuevo, se trata de definir un buffer para ir recorriendo el array de inodos, pasando cada vez un bloque a memoria principal desde el dispositivo virtual, e ir actualizándolo con esos inodos enlazados, para después salvarlo en el dispositivo mediante una llamada a la función *bwrite()*. Ese buffer será de tamaño BLOCKSIZE y tendrá la siguiente estructura de datos: struct inodo inodos[BLOCKSIZE/INODOSIZE],

Pasos:

- Primeramente leeremos el superbloque para obtener la localización del array de inodos.
- Habrá que inicializar el primer elemento del array de punteros directos de cada inodo con una variable incremental (ya que inicialmente todos los inodos están libres y en la lista enlazada cada uno apunta al siguiente). El último de la lista tendrá que apuntar a un nº muy grande (NULL), que podemos expresar con **UINT_MAX** (el máximo valor para un unsigned int) y en tal caso se requiere un #include <limits.h> en ficheros_basico.h.
- Iteraremos para cada bloque (desde la posición del 1er bloque del array de inodos hasta el último), y para cada inodo dentro de un bloque (cada bloque contiene una cantidad de inodos = (BLOCKSIZE / INODOSIZE), excepto el último bloque que no tiene porqué estar completamente lleno).

```
struct inodo inodos [BLOCKSIZE/INODOSIZE]
...
contInodos := SB.posPrimerInodoLibre+1;
//si hemos inicializado SB.posPrimerInodoLibre = 0
para (i=SB.posPrimerBloqueAI; i<=SB.posUltimoBloqueAI;i++) hacer
   para (j=0; j<BLOCKSIZE / INODOSIZE; j++) hacer
       inodos[j].tipo := 'l';  //libre
       si (contInodos < SB.totInodos) entonces
            inodos[j].punterosDirectos[0] := contInodos;
            contInodos++;
       si_no //hemos llegado al último inodo
           inodos[j].punterosDirectos[0] := UINT_MAX;
           //hay que salir del bucle, el último bloque no tiene por qué estar completo
       fsi
   fpara
   escribir el bloque de inodos en el dispositivo virtual
fpara
```

Indicamos que el tipo de inodo es libre ('l'). El resto de campos del inodo no es necesario inicializarlos.

Ahora hay que incorporar en **mi_mkfs.c** un include de ficheros_basico.h (en vez de bloques.h) y las **llamadas** a estas funciones (*initSB()*, *initMB()*, *initAI()*) para mejorar la creación del sistema de ficheros del nivel 1, y también la declaración de todas las funciones de este nivel en el fichero ficheros_basico.h.

En el nivel 3 restaremos los bloques ocupados por los metadatos (SB, MB y AI) a la cantidad de bloques libres del sistema.

COMPILACION

$gcc -o mi_mkfs mi_mkfs.c bloques.c ficheros_basico.c

o mucho mejor crear/ampliar un **Makefile.**

TESTS DE PRUEBA

Comenzar a desarrollar un programa de pruebas **leer_sf.c** que nos ayude a determinar la información almacenada en el superbloque, en el mapa de bits o en el array de inodos:. Sintaxis:

$ ./leer_sf <nombre_dispositivo>

(hay que montar y desmontar el dispositivo virtual y hacer un include de ficheros_basico.h!!!)

- De momento podéis mostrar por pantalla todos los campos del superbloque.
- Mostrar también el tamaño del struct inodo:

  printf ("sizeof struct inodo is: %**lu**\n", sizeof(struct inodo));
- Podéis hacer también un recorrido de la lista de inodos libres (mostrando para cada inodo el campo punterosDirectos[0]).

Más adelante podéis ampliar el programa para leer las otras estructuras de metadatos.

Ejemplo de ejecución de leer_sf para 100.000 bloques:

```
$ make clean
rm -rf *.o *~ mi_mkfs leer_sf
$ make
gcc -c -g -Wall -std=c99 -o bloques.o -c bloques.c
gcc -c -g -Wall -std=c99 -o ficheros_basico.o -c ficheros_basico.c
gcc -c -g -Wall -std=c99 -o mi_mkfs.o -c mi_mkfs.c
gcc -c -g -Wall -std=c99 -o leer_sf.o -c leer_sf.c
gcc bloques.o ficheros_basico.o  mi_mkfs.o -o mi_mkfs
gcc bloques.o ficheros_basico.o  leer_sf.o -o leer_sf
```

```
$ rm disco[8]
$ ./mi_mkfs disco 100000
$ ./leer_sf disco
DATOS DEL SUPERBLOQUE
posPrimerBloqueMB = 1
posUltimoBloqueMB = 13
posPrimerBloqueAI = 14
posUltimoBloqueAI = 3138
posPrimerBloqueDatos = 3139
posUltimoBloqueDatos = 99999
posInodoRaiz = 0
posPrimerInodoLibre = 0
cantBloquesLibres = 100000
cantInodosLibres = 25000
totBloques = 100000
totInodos = 25000

sizeof struct superbloque: 1024
sizeof struct inodo: 128


RECORRIDO LISTA ENLAZADA DE INODOS LIBRES
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28[9] …
… 24980 24981 24982 24983 24984 24985 24986 24987 24988 24989 24990 24991
24992 24993 24994 24995 24996 24997 24998 24999 -1
```

Anexo

La estructura del inodo es anónima, por lo que el C permite que uséis directamente los nombres de campos como inodo.punterosDirectos. Además, para no tener que estar definiendo las variables como "union inodo variable" se puede definir un tipo con typedef de nombre inodo_t (el “_t “ lo veis en muchos tipos estándares y es para indicar que es un "typedef"). Así ahora el código queda más breve y simple.

```
// Ejemplo uso de uniones
#include <stdlib.h>
#include <stdio.h>
typedef union _inodo {
  struct {
     //aquí irían los campos del inodo menos los punteros
```

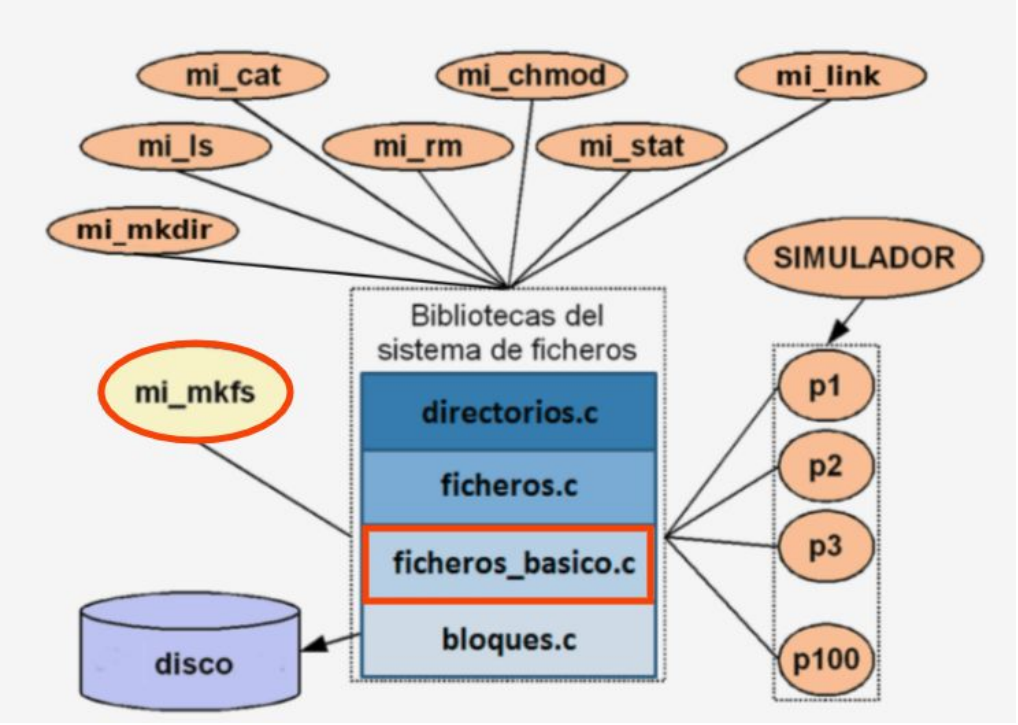
[8] Si ya existía previamente es preferible borrarlo

[9] Sólo he puesto los primeros y los últimos valores por cuestiones de espacio pero os debería mostrar la lista completa. Podéis redireccionar el comando a un fichero externo para ver mejor los resultados.

```
    unsigned int punterosDirectos[12];
    unsigned int punterosIndirectos[3];
  };
  char padding[INODOSIZE];
} inodo_t;

main() { //ejemplos de uso
  inodo_t inodo;
  inodo_t *ptr = &inodo; //se pueden usar también con punteros
  inodo.punterosDirectos[1] = 100;
  inodo.punterosIndirectos[2] = 101;
  ptr->punterosDirectos[2] = 102;
  printf("Size: %ld\n", sizeof(inodo_t));
}
```

Nivel 3: ficheros_basico.c {escribir_bit(), leer_bit(), reservar_bloque(), liberar_bloque(), escribir_inodo(), leer_inodo(), reservar_inodo()} y mi_mkfs.c

Continuemos con la definición de funciones básicas de gestión de ficheros (en **ficheros_basico.c**, y declaradas en su cabecera **ficheros_basico.h**), y actualizando **mi_mkfs.c** para formatear nuestro sistema de ficheros.



Hay que programar funciones básicas[1] de E/S para los bits del MB (en adelante MB):

1) int escribir_bit(unsigned int nbloque, unsigned int bit);

Esta función escribe el valor indicado por el parámetro bit: 0 (libre) ó 1 (ocupado) en un determinado bit del MB que representa el bloque nbloque.

Dado un nº de bloque físico, nbloque, del que queremos indicar si está libre o no, primeramente deberemos averiguar donde se ubica su bit correspondiente en el MB y luego en el dispositivo[2] (nº de bloque físico) para grabarlo cuando le hayamos dado el valor deseado.

Veámoslo paso a paso:

- Leer el superbloque para obtener la localización del MB

[1] Los parámetros indicados son orientativos. Si necesitáis adaptarlos lo hacéis, siempre y cuando las funciones hagan lo que se requiere. Igualmente podéis utilizar funciones auxiliarias cuando lo consideréis oportuno

[2] Recordemos que todas las operaciones de E/S con el dispositivo las hacemos por bloques

- Calculamos la posición del byte en el MB, posbyte, que contiene el bit que representa el nbloque y luego la posición del bit dentro de ese byte, posbit:

  posbyte = nbloque / 8

  posbit = nbloque % 8

- Hemos de determinar luego en qué bloque del MB, nbloqueMB, se halla ese bit para leerlo:

  nbloqueMB = posbyte/BLOCKSIZE

- Y finalmente hemos de obtener en qué posición absoluta del dispositivo virtual se encuentra ese bloque, nbloqueabs, donde leer/escribir el bit:

  nbloqueabs = nbloqueMB + SB.posPrimerBloqueMB

Veamos un ejemplo:

- nbloque = 40.003 (es el bloque que queremos indicar si está libre u ocupado, lo recibimos como parámetro)
- posbyte = nbloque / 8 = 5.000 (dividimos entre 8 porque los bits que representan los bloques físicos se agrupan de 8 en 8 para formar bytes, se trata de una división entera). Esto significa que el byte 5.000 del MB contiene el bit que representa el nbloque 40.003.
- posbit = nbloque % 8 = 40.003 % 8 = 3 (sería el resto de la división). Esto significa que el bit 3 (teniendo en cuenta que se empieza a contar desde el 0) del byte 5.000 del MB es el que representa el nbloque 40.003.
- nbloqueMB = posbyte / BLOCKSIZE = 5.000 / 1.024 = 4. Esto significa que el bloque 4 del MB, contando des de el 0 **de forma relativa en el MB**, contiene el byte 5.000 que a su vez contiene el bit 3 que representa al nbloque 40.003.
- nbloqueabs = SB.posPrimerBloqueMB + nbloqueMB = 1+4 = 5, es la posición absoluta del dispositivo donde se halla nbloqueMB, y la que emplearemos para realizar el bwrite()

Ahora que ya tenemos ubicado el bit en el dispositivo, leemos el bloque que lo contiene y cargamos el contenido en un buffer, bufferMB, en el que tendremos que modificar el bit deseado, pero preservando el valor de los demás bits del bloque.

Veámoslo paso a paso:

- Recordemos que posbyte es el byte que contiene el bit del del MB, y ese byte ahora lo tenemos contenido en bufferMB, que ocupa 1 bloque, así que necesitamos realizar la operación módulo con el tamaño de bloque para localizar su posición, y así quedará relativizado al valor de ese tamaño:

  posbyte = posbyte % BLOCKSIZE

  En el ejemplo anterior posbyte = 5.000. Para que nos sirva de índice en el buffer de tamaño 1024, hay que realizar el módulo con ese tamaño para obtener un valor dentro del rango, o sea que posbyte pasará a valer 5.000 % 1.024 = 904.

- Ahora que ya tenemos en memoria el byte, bufferMB[posbyte], podemos poner a 1 o a 0 el bit correspondiente. Para ello, primeramente, utilizaremos una máscara y realizaremos un desplazamiento de bits (tantos como indique el valor posbit) a la derecha:

```
unsigned char mascara = 128; // 10000000
```

```
mascara >>= posbit; // desplazamiento de bits a la derecha
```

- Para poner un bit a 1:

```
bufferMB[posbyte] |= mascara; // operador OR para bits
```

- Para poner un bit a 0:

```
bufferMB[posbyte] &= ~mascara; // operadores AND y NOT para bits
```

Veamos un ejemplo para posbit=3:

- mascara: 10000000
- Desplazando el 1er bit de la máscara a la derecha 3 posiciones ⇒ mascara: 00010000
- Para poner el bit a 1:

  Si hacemos el OR binario de la máscara con el byte del MB, obtendremos un 1 en la posición=3 y preservaremos el valor del resto:

  00010000 | xxx0xxxx = xxx1xxxx
  00010000 | xxx1xxxx = xxx1xxxx
- Para poner el bit a 0:

  Hacemos el NOT binario de la máscara ⇒ máscara: 11101111

  Si hacemos el AND binario de la máscara con el byte del MB, obtendremos un 0 en la posición=3 y preservaremos el valor del resto:

  11101111 & xxx0xxxx = xxx0xxxx
  11101111 & xxx1xxxx = xxx0xxxx

Por último escribimos ese buffer del MB en el dispositivo virtual con **bwrite()** en la posición que habíamos calculado anteriormente, nbloqueabs.

Ahora que ya disponemos de la función escribir_bit() podemos modificar la función **initMB()** para poner a 1 en el MB los bits que corresponden a los bloques que ocupa el superbloque, el propio MB, y el array de inodos[3]. Eso implicará también actualizar la cantidad de bloques libres en el superbloque y grabarlo.

2) char leer_bit(unsigned int nbloque);

[3]MEJORA OPTATIVA. Se podría optimizar sin hacer tantas llamadas a la función escribir_bit(), poniendo a 1 todos los bytes anteriores al que contiene el último bit y luego utilizar esta función sólo para poner a 1 los bits necesarios del último byte, o escribir directamente el byte resultante.

Por ejemplo, si los metadatos ocupasen 3139 bloques (tamSB+tamMB+tamAI), hemos de poner 3139 bits a 1, o sea que habría 392 bytes (3139/8) con todos sus bits a 1 (255 en decimal) y 1 byte adicional con los 3 restantes (3139%8=3). Esos bytes están todos en el bloque 1 del dispositivo virtual (ya que en un bloque caben 1024 bytes, suponiendo BLOCKSIZE=1KB). Por tanto llevando sólo ese primer bloque a memoria principal e iterando esos 392 bytes para igualarlos a 255 habremos puesto 3136 bits a 1 (392*8), y nos faltan 3 más, correspondientes al byte siguiente, el 392 (los anteriores iban del 0 al 391). Eso implica escribir en binario 11100000, o sea 224 en decimal ($2^7+2^6+2^5$), lo cual se tendría que expresar de forma generalizada. También habría que tener en cuenta que los bits correspondientes a los metadatos podrían ocupar más de 1 bloque, dependiendo de la cantidad total de bloques de nuestro dispositivo virtual y de BLOCKSIZE (ver test de prueba para comprobar que funciona para nbloques=1.000.000).

Lee un determinado bit del MB y devuelve el valor del bit leído.

Se procede gual que en la función anterior para obtener el byte del dispositivo que contiene el bit deseado y el bloque físico absoluto que lo contiene, pero en vez de escribir, lee el bit correspondiente utilizando un desplazamiento de bits a la derecha:

```
unsigned char mascara = 128;     // 10000000
mascara >>= posbit;              // desplazamiento de bits a la derecha
mascara &= bufferMB[posbyte];    // operador AND para bits
mascara >>= (7-posbit);          // desplazamiento de bits a la derecha
```

Veamos un ejemplo para posbit=3:

- mascara: 10000000
- Desplazando el 1er bit de la máscara a la derecha 3 ⇒ mascara: 00010000
- Si hacemos el AND binario de la máscara con el byte del MB, obtenemos el bit de la posición=3 y el resto queda a 0:

  00010000 & xxx0xxxx = 00000000
  00010000 & xxx1xxxx = 00010000
- En el byte resultado obtenido hacemos un desplazamiento de 7-posbit posiciones a la derecha, o sea de 4 y así nos queda:

  0000000**0** si originariamente había un 0 en el MB // 0 en decimal
  0000000**1** si originariamente había un 1 en el MB // 1 en decimal

---

En este nivel también hay que programar funciones básicas para reservar y liberar bloques:

3) int reservar_bloque();

Encuentra el primer bloque libre, consultando el MB, lo ocupa (con la ayuda de la función escribir_bit()) y devuelve su posición.

Veámoslo paso a paso:

- Comprobamos la variable del superbloque que nos indica si quedan bloques libres.
- Si aún quedan, hemos de localizar el 1er bloque libre del dispositivo virtual consultando cuál es el primer bit a 0 en el MB:
  - (1) Primero localizamos la posición del primer **bloque** del MB que tenga algún bit a 0, posBloqueMB y lo leemos:

    recorremos los bloques del MB (iterando con posBloqueMB) y los iremos cargando en bufferMB

    bread(posBloqueMB,bufferMB)

    hasta encontrar uno que tenga algún 0.

    Para ello utilizaremos un buffer auxiliar, bufferAux, inicializado a 1s:

    memset (bufferAux, 255, BLOCKSIZE)

y comparamos cada bloque leído del MB, bufferMB, con ese buffer auxiliar inicializado a 1s, utilizando la función memcmp().

- (2) Luego localizamos qué **byte** dentro de ese bloque tiene algún 0:

  Cuando salgamos de la iteración, en bufferMB estará el bloque que contiene al menos un 0 y buscamos en ese bloque, procedente del MB, la posición del primer byte, posbyte, que tenga algún 0 (podemos hacerlo recorriendo ese bloque y comparando cada byte con 255).

- (3) Finalmente localizamos el primer **bit** dentro de ese byte que vale 0:

  Buscamos en ese byte, bufferMB[posbyte], en qué posición, posbit, está el 0, empezando por la izquierda:

```
unsigned char mascara = 128; // 10000000
int posbit = 0;
while (bufferMB[posbyte] & mascara) {
  posbit++;
  bufferMB[posbyte] <<= 1; // desplaz. de bits a la izqda
}
```

  Veamos un ejemplo para un byte con valor 251 (11111011 en binario):

  – mascara: 10000000, posbit = 0

  – Iteramos un AND binario del byte del MB con la máscara, incrementamos el contador y desplazamos un bit a la izquierda:

```
11111011 & 10000000 = 10000000, posbit = 1
11110110 & 10000000 = 10000000, posbit = 2
11101100 & 10000000 = 10000000, posbit = 3
11011000 & 10000000 = 10000000, posbit = 4
10110000 & 10000000 = 10000000, posbit = 5
01100000 & 10000000 = 00000000 //fin
```

- Para determinar cuál es finalmente el nº de bloque (nbloque) que podemos reservar (posición absoluta del dispositivo), necesitaremos efectuar el siguiente cálculo:

```
nbloque = ((posBloqueMB - SB.posPrimerBloqueMB) * BLOCKSIZE+
                    posbyte) * 8 + posbit;
```

- Utilizamos la función escribit_bit() pasándole como parámetro ese nº de bloque y un 1 para indicar que el bloque está reservado.
- Decrementamos la cantidad de **bloques libres** en el campo correspondiente del superbloque, y salvamos el superbloque
- Grabamos un buffer de 0s en la posición del nbloque del dispositivo por si había basura (podría tratarse de un bloque reutilizado por el sistema de ficheros)
- Devolvemos el nº de bloque que hemos reservado, nbloque

4) int liberar_bloque(unsigned int nbloque);

Libera un bloque determinado (con la ayuda de la función escribir_bit()).

Veámoslo paso a paso:

- Ponemos a 0 el bit del MB correspondiente al bloque nbloque (lo recibimos como parámetro)
- Incrementamos la cantidad de **bloques libres** en el superbloque, pero no limpiamos el bloque en la zona de datos; se queda basura pero se interpreta como espacio libre. Salvamos el superbloque.
- Devolvemos el nº de bloque liberado, nbloque.

Hay que programar también funciones básicas para escribir y leer inodos:

5) int escribir_inodo(unsigned int ninodo, struct inodo inodo);

Escribe el contenido de una variable de tipo struct inodo en un determinado inodo del array de inodos, inodos.

Observación: como la escritura se hace por bloques, hay que preservar el valor de los demás inodos del bloque.

Veámoslo paso a paso:

- Leemos el superbloque para obtener la localización del array de inodos
- Obtenemos el nº de bloque del array de inodos que tiene el inodo solicitado
- Empleamos un array de inodos, del tamaño de la cantidad de inodos que caben en un bloque: struct inodo inodos[BLOCKSIZE/INODOSIZE], como buffer de lectura del bloque que hemos de leer
- Una vez que tenemos el bloque en memoria escribimos el inodo en el lugar correspondiente del array: ninodo%(BLOCKSIZE/INODOSIZE)
- El bloque modificado lo escribimos en el dispositivo virtual utilizando la función bwrite()

6) int leer_inodo(unsigned int ninodo, struct inodo *inodo);

Lee un determinado inodo del array de inodos para volcarlo en una variable de tipo struct inodo pasada por referencia.

Veámoslo paso a paso:

- Leemos el superbloque para obtener la localización del array de inodos
- Obtenemos el nº de bloque del array de inodos que tiene el inodo solicitado
- Empleamos un array de inodos, del tamaño de la cantidad de inodos que caben en un bloque: struct inodo inodos[BLOCKSIZE/INODOSIZE], como buffer de lectura del bloque que hemos de leer
- El inodo solicitado está en la posición ninodo%(BLOCKSIZE/INODOSIZE) del buffer

- Si ha ido todo bien devolvemos 0

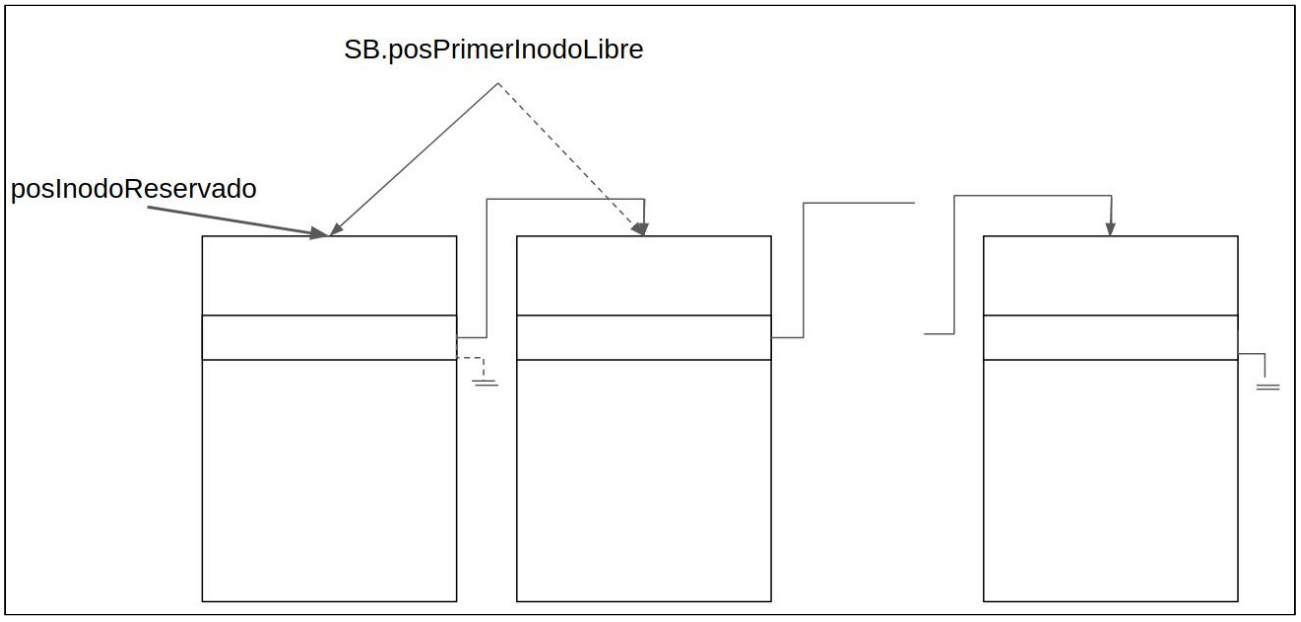
Finalmente nos queda programar funciones básicas para reservar y liberar inodos (ésta última la dejaremos para el Nivel 5):

7) int reservar_inodo(unsigned char tipo, unsigned char permisos);

Encuentra el primer inodo libre (dato almacenado en el superbloque), lo reserva (con la ayuda de la función escribir_inodo()), **devuelve su número** y actualiza la lista enlazada de inodos libres.

Veámoslo paso a paso:

- Comprobar si hay inodos libres y si no hay inodos libres indicar error y salir
- Primeramente **actualizar la lista enlazada de inodos libres** de tal manera que el superbloque apunte al siguiente de la lista. Tendremos la precaución de guardar en una variable auxiliar posInodoReservado cual era el primer inodo libre, ya que éste es el que hemos de devolver.



Lista enlazada de nodos libres antes y después de reservar un inodo

- A continuación inicializamos todos los campos del inodo al que apuntaba inicialmente el superbloque:
  - tipo (pasado como argumento)
  - permisos (pasados como argumento)
  - cantidad de enlaces de entradas en directorio: 1
  - tamaño en bytes lógicos: 0
  - *timestamp* de creación para todos los campos de fecha y hora: time(NULL)
  - cantidad de bloques ocupados en la zona de datos: 0
  - punteros a bloques directos: 0 (el valor 0 indica que no apuntan a nada)
  - punteros a bloques indirectos. 0 (el valor 0 indica que no apuntan a nada)
- Utilizar la función escribir_inodo() para escribir el inodo inicializado en la posición del que era el primer inodo libre, posInodoReservado.

- Actualizar la cantidad de inodos libres, y reescribir el superbloque.
- **Devolver posInodoReservado.**

En el programa **mi_mkfs.c** habrá que **crear el directorio raíz.** Podemos utilizar la función reservar_inodo ('d', 7) para ello.

En la inicialización del superbloque tendremos que haber indicado que el primer inodo libre es el 0 y que la cantidad de inodos libres inicial es ninodos (la función reservar_inodo() actualizará esos valores).

Tras la creación del directorio raíz, el primer inodo libre pasará a ser el 1 y en el sistema habrá un inodo libre menos.

TESTS DE PRUEBA

Para comprobar el buen funcionamiento de las funciones de este nivel podéis modificar el programa de pruebas leer_sf.c[4] para:

- mostrar el superbloque (ya se habrán inicializado los metadatos y habrán esos bloques libres menos, y también se habrá creado el inodo raíz con lo cual habrá 1 inodo libre menos y se habrá actualizado la cabecera de la lista de inodos libres)
- mostrar el MB (y así comprobar el funcionamiento de escribir_bit() y leer_bit()). Si no queréis mostrar los 100.000 bits bastan el 1º y último de cada zona.
- reservar y liberar un bloque (y así comprobar las funciones reservar_bloque() y liberar_bloque()). Para mostrar los cambios en la cantidad de bloques libres tras cada acción habra que leer el superbloque.
- mostrar el inodo del directorio raíz (y así comprobar reservar_inodo(), escribir_inodo() y leer_inodo()).

Para mostrar en formato amigable los sellos de tiempo de un inodo que están en epoch:

```
#include <time.h> //esta librería incluirla en ficheros_basico.h
...
struct tm *ts;
char atime[80];
char mtime[80];
char ctime[80];

struct inodo inodo;
int ninodo;
...
leer_inodo(ninodo, &inodo);
ts = localtime(&inodo.atime);
strftime(atime, sizeof(atime), "%a %Y-%m-%d %H:%M:%S", ts);
ts = localtime(&inodo.mtime);
strftime(mtime, sizeof(mtime), "%a %Y-%m-%d %H:%M:%S", ts);
ts = localtime(&inodo.ctime);
strftime(ctime, sizeof(ctime), "%a %Y-%m-%d %H:%M:%S", ts);
printf("ID: %d ATIME: %s MTIME: %s CTIME: %s\n",ninodo,atime,mtime,ctime);
...
```

Ejemplo de ejecución de leer_sf en este nivel para 100.000 bloques y para 1.000.000 con BLOCKSIZE=1KB:

```
$ ./mi_mkfs disco 100000
$ ./leer_sf disco
DATOS DEL SUPERBLOQUE
posPrimerBloqueMB = 1
posUltimoBloqueMB = 13
posPrimerBloqueAI = 14
posUltimoBloqueAI = 3138
```

[4] En este nivel ya no hay que mostrar la inicialización de la lista enlazada de inodos

posPrimerBloqueDatos = 3139
posUltimoBloqueDatos = 99999
posInodoRaiz = 0
posPrimerInodoLibre = 1
cantBloquesLibres = 96861
cantInodosLibres = 24999
totBloques = 100000
totInodos = 25000

RESERVAMOS UN BLOQUE Y LUEGO LO LIBERAMOS
Se ha reservado el bloque físico nº 3139 que era el 1º libre indicado por el MB.
SB.cantBloquesLibres = 96860
Liberamos ese bloque y después SB.cantBloquesLibres = 96861.

MAPA DE BITS CON BLOQUES DE METADATOS OCUPADOS[5]

[leer_bit()→ nbloque: 0, posbyte:0, posbit:0, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posSB (o sea al BF nº 0) = 1

[leer_bit()→ nbloque: 1, posbyte:0, posbit:1, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posPrimerBloqueMB (o sea al BF nº 1) = 1

[leer_bit()→ nbloque: 13, posbyte:1, posbit:5, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posUltimoBloqueMB (o sea al BF nº 13) = 1

[leer_bit()→ nbloque: 14, posbyte:1, posbit:6, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posPrimerBloqueAI (o sea al BF nº 14) = 1

[leer_bit()→ nbloque: 3138, posbyte:392, posbit:2, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posUltimoBloqueAI (o sea al BF nº 3138) = 1

[leer_bit()→ nbloque: 3139, posbyte:392, posbit:3, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posPrimerBloqueDatos (o sea al BF nº 3139) = 0

[leer_bit()→ nbloque: 99999, posbyte:12499[6], posbit:7, nbloqueMB:12,
nbloqueabs:13)]
valor del bit correspondiente a posUltimoBloqueDatos (o sea al BF nº 99999) = 0

DATOS DEL DIRECTORIO RAIZ
tipo: d
permisos: 7
atime: Fri 2020-03-06 10:55:15
ctime: Fri 2020-03-06 10:55:15
mtime: Fri 2020-03-06 10:55:15

[5] Por simplicidad basta mostrar los bits de los bloques de inicio y fin de cada zona del dispositivo en vez de un listado de los nbloques del dispositivo
[6] Si lo mostráis después de normalizar, el valor de posbyte dará 12499 % 1024 = 211

```
nlinks: 1
tamEnBytesLog: 0
numBloquesOcupados: 0
```

```
$ rm disco
$ ./mi_mkfs disco 1000000 [7]
$ ./leer_sf disco
DATOS DEL SUPERBLOQUE
posPrimerBloqueMB = 1
posUltimoBloqueMB = 123
posPrimerBloqueAI = 124
posUltimoBloqueAI = 31373
posPrimerBloqueDatos = 31374
posUltimoBloqueDatos = 999999
posInodoRaiz = 0
posPrimerInodoLibre = 1
cantBloquesLibres = 968626
cantInodosLibres = 249999
totBloques = 1000000
totInodos = 250000

RESERVAMOS UN BLOQUE Y LUEGO LO LIBERAMOS
Se ha reservado el bloque físico nº 31374 que era el 1º libre indicado por el MB
SB.cantBloquesLibres = 968625
Liberamos ese bloque y después SB.cantBloquesLibres = 968626

MAPA DE BITS CON BLOQUES DE METADATOS OCUPADOS
[leer_bit(0)→ posbyte:0, posbit:0, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posSB (o sea al BF nº 0)  = 1

[leer_bit(1)→ posbyte:0, posbit:1, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posPrimerBloqueMB (o sea al BF nº 1) = 1

[leer_bit(123)→ posbyte:15, posbit:3, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posUltimoBloqueMB (o sea al BF nº 123) = 1

[leer_bit(124)→ posbyte:15, posbit:4, nbloqueMB:0, nbloqueabs:1)]
valor del bit correspondiente a posPrimerBloqueAI (o sea al BF nº 124) = 1

[leer_bit(31373)→ posbyte:3921, posbit:5, nbloqueMB:3, nbloqueabs:4)]
valor del bit correspondiente a posUltimoBloqueAI (o sea al BF nº 31373) = 1

[leer_bit(31374)→ posbyte:3921, posbit:6, nbloqueMB:3, nbloqueabs:4)]
valor del bit correspondiente a posPrimerBloqueDatos (o sea al BF nº 31374) = 0

[leer_bit(999999)→ posbyte:124999, posbit:7, nbloqueMB:122, nbloqueabs:123)]
```

[7] Ahora los bits del MB relativos a los metadatos ocuparán más de 1 bloque

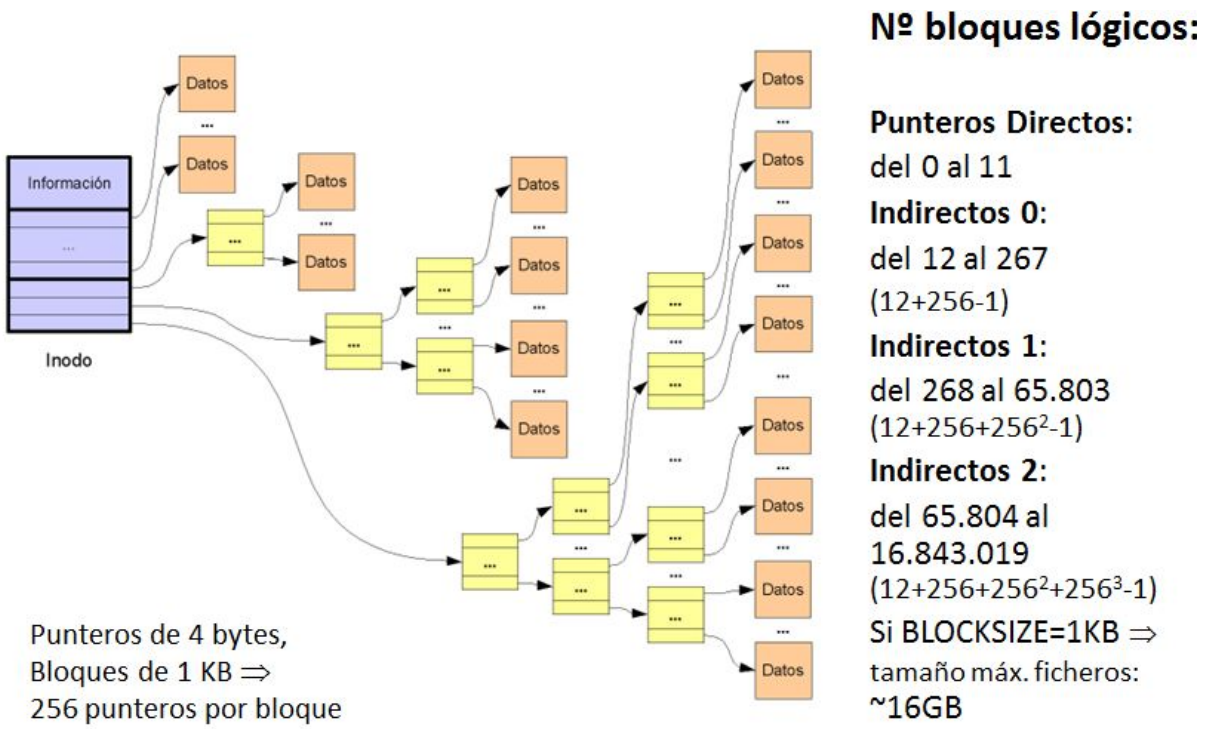
valor del bit correspondiente a posUltimoBloqueDatos (o sea al BF nº 999999) = 0

DATOS DEL DIRECTORIO RAIZ

tipo: d
permisos: 7
atime: Fri 2020-03-06 10:55:35
ctime: Fri 2020-03-06 10:55:35
mtime: Fri 2020-03-06 10:55:35
nlinks: 1
tamEnBytesLog: 0
numBloquesOcupados: 0

Nivel 4: ficheros_basico.c {traducir_bloque_inodo() y auxiliares: obtener_indice() y obtener_nrangoBL()}

Antes de nada, repasemos cómo los inodos apuntan a los bloques de datos:



En concreto, tenemos inodos con 12 punteros a bloques directos y 3 punteros indirectos, de diferentes niveles, que apuntan a bloques de índices, y suponemos que el tamaño de bloque es de 1.024 bytes (por lo que en un bloque caben 256 punteros ya que cada puntero se representa con 4 bytes correspondientes al sizeof(unsigned int)). Entonces:

- Los punteros directos a bloques de datos permiten encontrar los primeros 12 bloques lógicos del inodo, es decir, los comprendidos entre el 0 y el 0+12-1: o sea del 0 al 11.
- El puntero de bloques indirectos 0 permiten encontrar los siguientes 256 bloques lógicos del inodo, es decir, los comprendidos entre el 0+12 y el 0+12+256-1: o sea del 12 al 267.
- El puntero de bloques indirectos 1 permiten encontrar los siguientes 256^2 (65.536) bloques lógicos del inodo, es decir, los comprendidos entre el 0+12+256 y el $0+12+256+256^2-1$: o sea del 268 al 65.803.
- El puntero de bloques indirectos 2 permitenencontrar los siguientes 256^3 (16.777.216) bloques lógicos del inodo, es decir, los comprendidos entre el $0+12+256+256^2$ y el $0+12+256+256^2+256^3-1$: o sea del 65.804 al 16.843.019.

A partir de los datos anteriores podemos crear una función auxiliar **obtener_nRangoBL()** que me asocie un nivel a cada rango de bloques lógicos, devolviendo el nivel del bloque lógico indicado, siendo el nivel 0 para [0 , 11], 1 para [12 , 267], 2 para [268 , 65.803] y 3 para [65.804 , 16.843.019]. También ha de actualizar una variable puntero para que apunte donde lo hace el puntero correspondiente del inodo.

Primeramente podemos definir constantes simbólicas (en ficheros_basico.h) que nos ayuden a determinar los rangos de punteros:

```
#define NPUNTEROS (BLOCKSIZE/sizeof(unsigned int)) //256
#define DIRECTOS 12
#define INDIRECTOS0 (NPUNTEROS + DIRECTOS)        //268
#define INDIRECTOS1 (NPUNTEROS * NPUNTEROS + INDIRECTOS0)   //65.804
#define INDIRECTOS2 (NPUNTEROS * NPUNTEROS * NPUNTEROS + INDIRECTOS1)
//16.843.020
//cuidado con los paréntesis !!!
```

Y luego (en ficheros_basico.c) definimos la función obtener_rangoL() para obtener el rango de punteros en el que se sitúa el bloque lógico que buscamos (0:D, 1:I0, 2:I1, 3:I2), y obtenemos además la dirección almacenada en el puntero correspondiente del inodo:

```
funcion obtener_nrangoBL (inodo: struct inodo, nblogico:unsigned ent, *ptr:unsigned
ent) devolver nrangoBL:ent
        si nblogico<DIRECTOS entonces
          *ptr:=inodo.punterosDirectos[nblogico]
          devolver 0
        si_no si nblogico<INDIRECTOS0 entonces
          *ptr:=inodo.punterosIndirectos[0]
          devolver 1
        si_no si nblogico<INDIRECTOS1 entonces
          *ptr:=inodo.punterosIndirectos[1]
          devolver 2
        si_no si nblogico<INDIRECTOS2 entonces
          *ptr:=inodo.punterosIndirectos[2]
          devolver 3
        si_no
          *ptr:=0
          error("Bloque lógico fuera de rango")
          devolver -1
        fsi
ffuncion
```

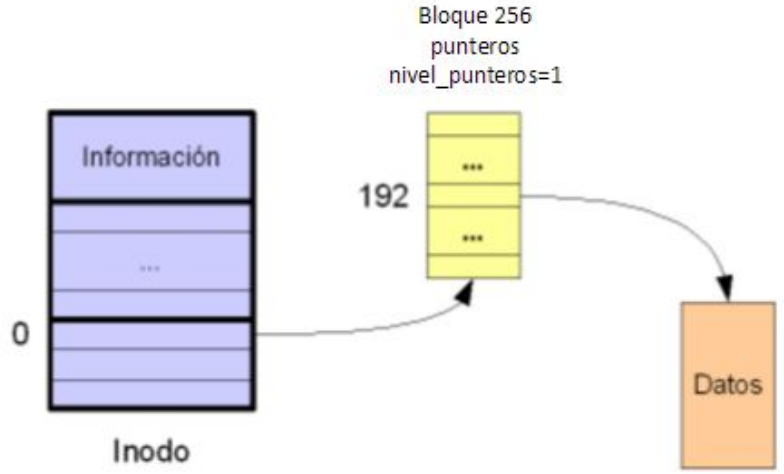
Veamos ahora ejemplos de cómo se obtienen los índices de los bloques de punteros:

Si, por ejemplo, queremos traducir el **bloque lógico número 204**:
Como 12 <= 204 <= 267, hemos de recurrir al punterosIndirectos[0].

Restamos los 12 punteros anteriores para obtener un valor índice dentro del rango de los 256 punteros, [0,255], del bloque de índices de nivel 1 que cuelga de punterosIndirectos[0]:

204 - 12 = **192** (⊂[0,255])

Por tanto el número de nuestro bloque físico se encuentra en el puntero número **192** del bloque de índices de nivel 1 apuntado por punterosIndirectos[0].



Si, por ejemplo, queremos traducir el **bloque lógico número 30.004**:

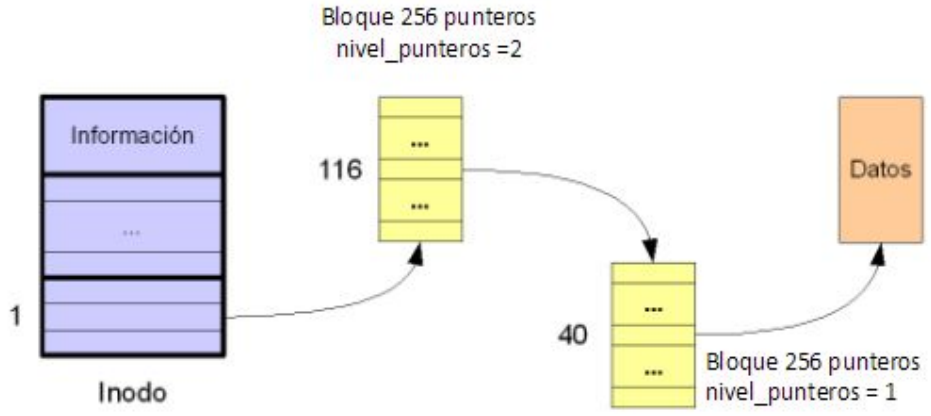
Como 268 <= 30.004 <= 65.803, hemos de recurrir a punterosIndirectos[1]. Restamos los 268 punteros anteriores (12+256) para obtener un valor dentro del rango de los 256² (65.536) bloques que puedo direccionar con la estructura global de punteros que cuelga de punterosIndirectos[1]:

30.004 - 268 = **29.736** (⊂[0, 65.535])

Dado que ese valor puede estar en uno de los 256 bloques de punteros de nivel 1, que cuelgan del bloque de punteros de nivel 2, tendremos que dividir ese valor entre 256 para obtener de cuál se trata (índice en el bloque de punteros de nivel 2). El resto de la división (módulo) nos dará el índice correspondiente en ese bloque de punteros de nivel 1, apuntado por el que hemos obtenido en el nivel 2:

29.736 / 256 = **116**
29.736 % 256 = **40**

Por tanto, el número de nuestro bloque físico se encuentra en el puntero número **40** del bloque apuntado por el puntero número **116** del bloque apuntado por punterosIndirectos[1].



Si, por ejemplo, queremos traducir el **bloque lógico número 400.004**:

Como 65.804 <= 400.004 <= 16.843.019, hemos de recurrir a punterosIndirectos[2].

Restamos los 65.804 punteros anteriores (12+256+65.536) para obtener un valor dentro del rango de los 256^3 (16.777.216) bloques que puedo direccionar con la estructura global de punteros que cuelga de punterosIndirectos[2]:

$$400.004 - 65.804 = 334.200 \ (\subset [0, 16.777.215])$$

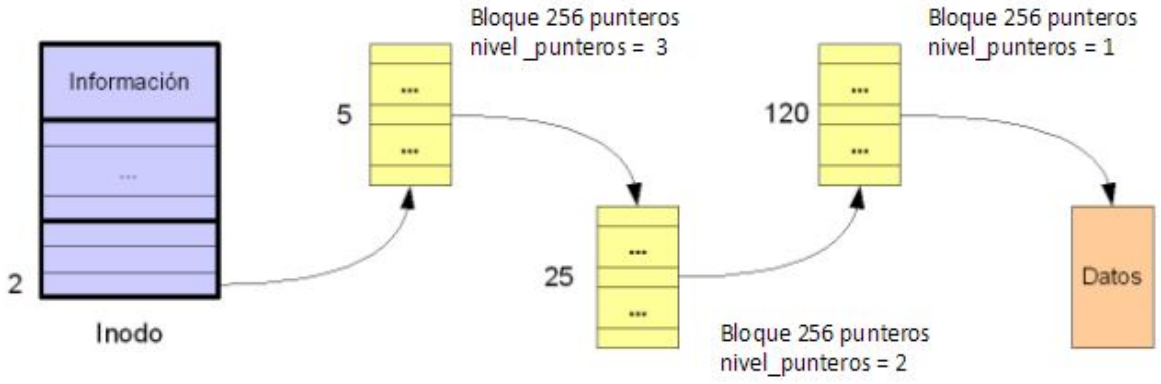
Para saber ese bloque a qué índice de nivel 3 corresponde, tendremos que dividirlo entre los 256^2 que salen de su estructura:

$$334.200 / 65.536 = 5$$

El resto de esa división habrá que dividirlo entre los 256 bloques de punteros que cuelgan para determinar el índice del bloque de punteros de nivel 2, y el módulo nos dará el índice del nivel 1:

$$334.200 \ \% \ 65.536 = 6.520$$
$$6.520 / 256 = 25$$
$$6.520 \ \% \ 256 = 120$$

Por tanto, el número de nuestro bloque físico se encuentra en el puntero número **120** del bloque apuntado por el puntero número **25** del bloque apuntado por el puntero número **5** del bloque apuntado por indirectos 2.



De los ejemplos anteriores podemos deducir una función, **obtener_indice()** para generalizar la obtención de los índices de los bloques de punteros.

```
funcion obtener_indice (nblogico: ent, nivel_punteros:ent) devolver ind:ent
si nblogico<DIRECTOS entonces devolver nblogico   //ej nblogico=8
si_no si nblogico<INDIRECTOS0 entonces devolver nblogico-DIRECTOS   //ej nblogico=204
si_no si nblogico<INDIRECTOS1 entonces   //ej nblogico=30.004
  si nivel_punteros=2 entonces
    devolver (nblogico-INDIRECTOS0) / NPUNTEROS
  si_no si nivel_punteros=1 entonces
    devolver (nblogico-INDIRECTOS0) % NPUNTEROS
  fsi
si_no si nblogico<INDIRECTOS2 entonces   //ej nblogico=400.004
  si nivel_punteros=3 entonces
```

```
    devolver (nblogico-INDIRECTOS1)/(NPUNTEROS*NPUNTEROS)
  si_no si nivel_punteros=2 entonces
    devolver ((nblogico-INDIRECTOS1)%(NPUNTEROS*NPUNTEROS)) / NPUNTEROS
  si_no si nivel_punteros=1 entonces
    devolver ((nblogico-INDIRECTOS1)%(NPUNTEROS*NPUNTEROS)) % NPUNTEROS
  fsi
fsi
ffuncion
```

Ahora ya podemos desarrollar la función de **traducir_bloque_inodo()**:

1) int traducir_bloque_inodo(unsigned int *ninodo*, unsigned int *nblogico*, char *reservar*);

Esta función se encarga de obtener el nº de bloque físico correspondiente a un bloque lógico determinado del inodo indicado. Enmascara la gestión de los diferentes rangos de punteros directos e indirectos del inodo, de manera que funciones externas no tienen que preocuparse de cómo acceder a los bloques físicos apuntados desde el inodo.

Para desarrollarla, se puede optar tanto por una versión recursiva como por una versión iterativa y utilizar las funciones auxiliares para obtener el nº de rango de punteros, el puntero correspondiente del inodo y los índices de los bloques de punteros.

La misma función nos puede servir tanto para sólo consultar (caso del mi_read_f()) como para consultar y reservar un bloque libre si ningún bloque físico es apuntado por el número de bloque lógico (caso del mi_write_f()).

Podemos indicar este comportamiento con el argumento reservar. De manera genérica:

- Si reservar vale 0 utilizaremos traducir_bloque_inodo() únicamente para consultar:
  - Si existe bloque físico de datos, la función devolverá su posición.
  - Si no existe bloque físico de datos, dará error.
- Si reservar vale 1 utilizaremos traducir_bloque_inodo() para consultar y, si no existe bloque físico, también para reservar:
  - Si existe bloque físico de datos, la función devolverá su posición.
  - Si no existe bloque físico de datos, lo reservará y se devolverá su posición

Hay que tener en cuenta si existen o no los bloques de punteros intermedios que precisemos atravesar hasta llegar al bloque de datos. En caso de que alguno/s no exista/n, si estamos en modo de lectura, avisaremos del error, y si estamos en modo escritura, habrá que reservarlos (inicializando previamente un buffer con 0s para indicar que no apuntan a nada) y enlazarlos adecuadamente. Cada vez que reservemos un bloque (sea para datos o para punteros) habrá que incrementar el campo de número de bloques ocupados del inodo.

Si se alteran los punteros del inodo y/o la cantidad de bloques ocupados hay que actualizar el ctime!

Un ejemplo de pseudocódigo para la función optimizada sería:

```
funcion traducir_bloque_inodo(ninodo:ent, nblogico:ent, reservar: char) devolver ptr:ent
  var
    inodo: estructura inodo
    ptr, ptr_ant, salvar_inodo, nRangoBL, nivel_punteros, indice: ent
    buffer[NPUNTEROS]: ent
  fvar
  leer_inodo (ninodo, &inodo)
  ptr := 0, ptr_ant := 0, salvar_inodo := 0
  nRangoBL := obtener_nRangoBL(inodo, nblogico, &ptr); //0:D, 1:I0, 2:I1, 3:I2
  nivel_punteros := nRangoBL //el nivel_punteros +alto es el que cuelga del inodo
  mientras nivel_punteros>0 hacer //iterar para cada nivel de indirectos
    si ptr=0 entonces //no cuelgan bloques de punteros
      si reservar=0 entonces devolver -1 //error lectura bloque inexistente
      si_no //reservar bloques punteros y crear enlaces desde inodo hasta datos
        salvar_inodo := 1
        ptr := reservar_bloque() //de punteros
        inodo.numBloquesOcupados++
        inodo.ctime = time(NULL) //fecha actual
        si nivel_punteros = nRangoBL entonces
          //el bloque cuelga directamente del inodo
          inodo.punterosIndirectos[nRangoBL-1] := ptr // (imprimirlo para test)
        si_no   //el bloque cuelga de otro bloque de punteros
          buffer[indice] := ptr // (imprimirlo para test)
          bwrite(ptr_ant, buffer)
        fsi
      fsi
    fsi
    bread(ptr, buffer)
    indice := obtener_indice(nblogico, nivel_punteros)
    ptr_ant := ptr //guardamos el puntero
    ptr := buffer[indice] // y lo desplazamos al siguiente nivel
    nivel_punteros--
  fmientras   //al salir de este bucle ya estamos al nivel de datos

  si ptr=0 //no existe bloque de datos
    si reservar=0 entonces devolver -1  //error lectura ∄ bloque
    si_no
      salvar_inodo := 1
      ptr = reservar_bloque() //de datos
      inodo.numBloquesOcupados++
      inodo.ctime = time(NULL)
      si nRangoBL=0 entonces
        inodo.punterosDirectos[nblogico] := ptr // (imprimirlo para test)
      si_no
        buffer[indice] := ptr // (imprimirlo para test)
        bwrite(ptr_ant, buffer)
      fsi
    fsi
  fsi
```

```
  si salvar_inodo=1 entonces
    escribir_inodo(ninodo, inodo)  //sólo si lo hemos actualizado
  fsi
  devolver ptr //nbfisico del bloque de datos
ffuncion
```

Recomendación:

- Hacer el seguimiento a mano del algoritmo (con reservar=1) para los siguientes bloques lógicos de un mismo inodo: 8, 204, 30.004, 400.004, y 468.750, suponiendo que estamos en la situación inicial de que todos los punteros apuntan a 0. Comprobar los índices obtenidos para cada nivel con los ejemplos proporcionados, y observar cómo se va creando la estructura multinivel de punteros.

TESTS DE PRUEBA

En este nivel leer_sf.c ya no tiene que reservar y liberar un bloque, ni mostrar el mapa de bits ni el directorio raíz (no borréis las sentencias, tan sólo ponerlas como comentarios). Ahora habrá que añadir a **leer_sf.c** las instrucciones necesarias para comprobar la traducción de bloques de diferentes rangos de punteros.

Para ello necesitaremos previamente reservar un inodo con la función reservar_inodo(), y llamar a la función traducir_bloque_inodo() para traducir los bloques lógicos siguientes: 8, 204, 30.004, 400.004 y 468.750

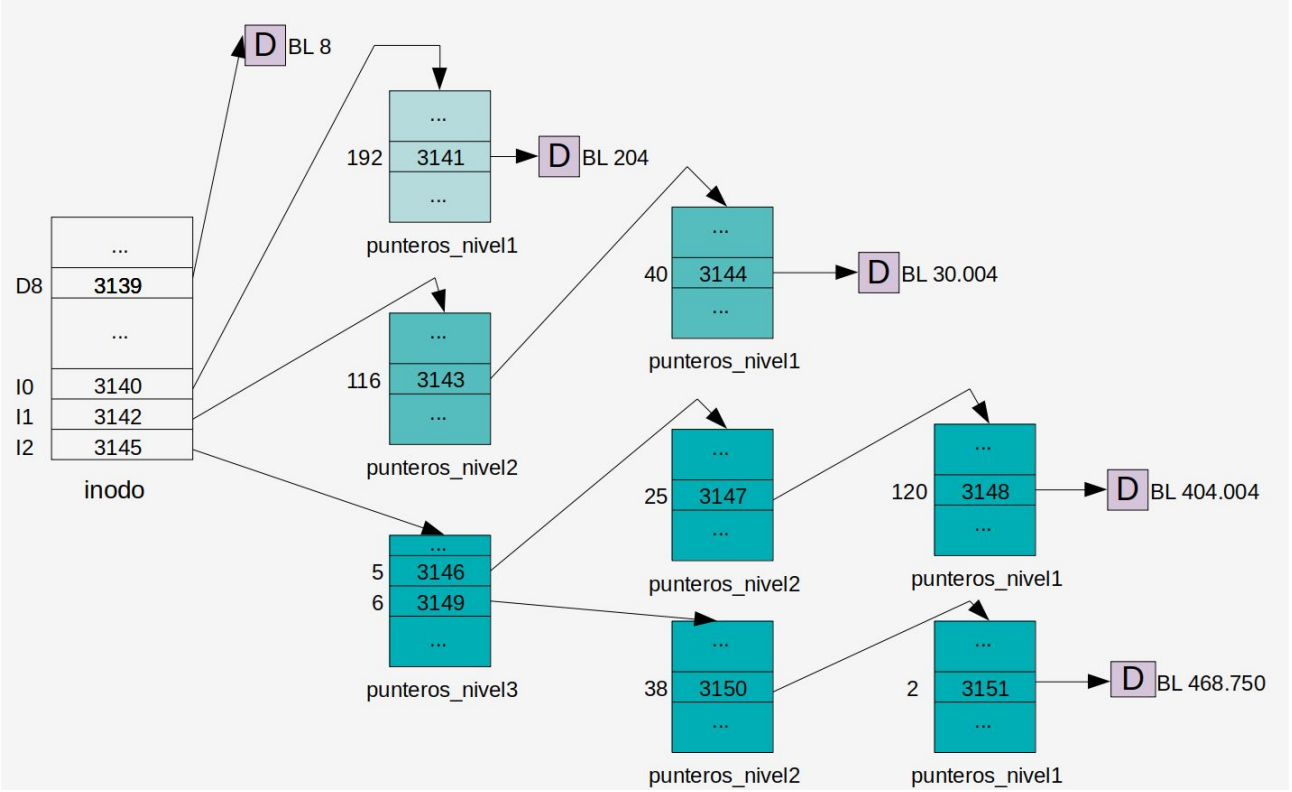
En la función traducir_bloque_inodo() de ficheros_basico.c poner los fprintf necesarios, justo después de que el valor de ptr sea asignado a alguna variable (en las 4 situaciones posibles) para mostrar:

- el nivel de punteros
- el índice para ese nivel
- y los bloques físicos reservados para bloques de punteros y bloques de datos

```
$ ./mi_mkfs disco 100000
$ ./leer_sf disco
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

INODO 1. TRADUCCION DE LOS BLOQUES LOGICOS 8, 204, 30.004, 400.004 y 468.750

[traducir_bloque_inodo()→ inodo.punterosDirectos[8] = 3139 (reservado BF 3139 para BL 8)]
[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)]
[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)]
[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3145 (reservado BF 3145 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)]
[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)]

DATOS DEL INODO RESERVADO 1
tipo: f
permisos: 6
atime: Wed 2020-03-25 16:50:21
ctime: Wed 2020-03-25 16:50:21
mtime: Wed 2020-03-25 16:50:21
nlinks: 1
tamEnBytesLog: 0
numBloquesOcupados [1]: 13

SB.posPrimerInodoLibre = 2
```

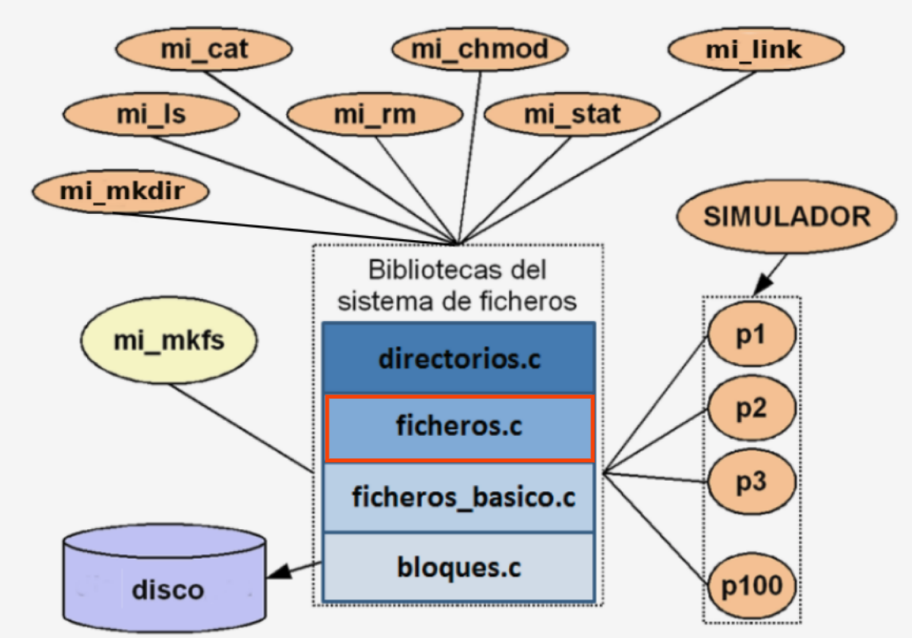
[1] Los ha tenido que crear porque no existían pero no contienen nada. Para hacer esta prueba estamos llamando a traducir_bloque_inodo() con reservar=1 para ver que crea bien los punteros. Después borraremos en leer_sf.c esas llamadas ya que en la práctica sólo tiene sentido llamar a traducir_bloque_inodo() con reservar=1 cuando vayamos a escribir contenido en los ficheros o a crear entradas en los directorios, ambas acciones desde una capa superior de nuestra biblioteca mediante la función mi_write_f().

El siguiente gráfico ilustra esta ejecución:



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

Construiremos el programa **ficheros.c** de nuestra biblioteca de funciones (en este punto un fichero está únicamente identificado por el número de su inodo)



En este nivel desarrollaremos las funciones que nos permitan escribir y leer bytes en un fichero (identificado por su nº de inodo) y también cambiar los permisos.

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): 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: (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 donde hay que escribir:
  primerBLogico=offset/BLOCKSIZE
- Calculamos cuál va a ser el último bloque lógico donde hay que escribir:
  ultimoBLogico=(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 en el bloque para el offset:
  desp1 = offset % BLOCKSIZE
- Calculamos el desplazamiento 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 primer y último bloque coincidan** (primerBLogico==ultimoBLogico), y por tanto el buffer que vayamos 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, correspondiente a primerBLogico, 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 (nbytes < BLOCKSIZE), o lo que es lo mismo desp2-desp1+1 bytes, del buf_original en la posición buf_bloque+desp1
- Escribimos buf_bloque modificado en el nº de bloque físico correspondiente

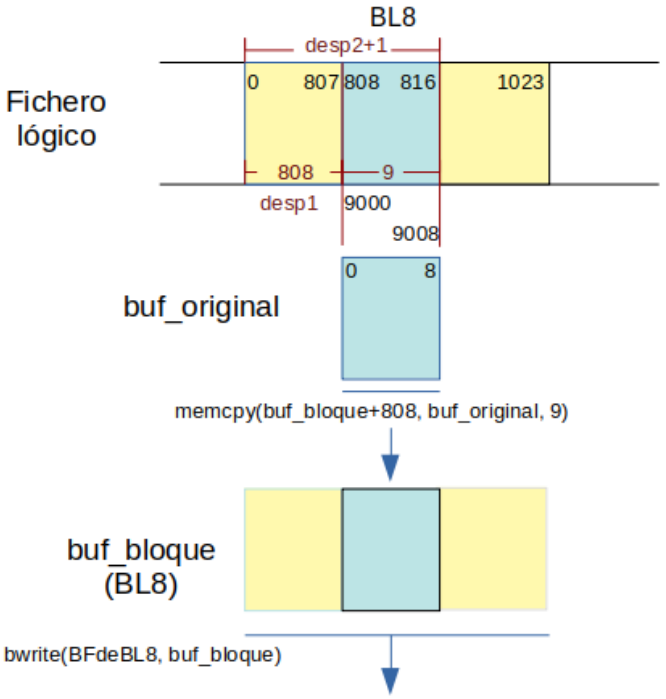
Para entenderlo mejor, veamos un ejemplo en el que queremos escribir 9 bytes (almacenados en el buffer buf_original) a partir del byte lógico número 9000 del inodo (offset). Suponemos que el tamaño de bloque es de 1024 bytes:

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

Ejemplo para offset (byte lógico inicial) = 9000 y nbytes = 9 con BLOCKSIZE=1024

9000/1024=BL8
9000%1024= 808 (desp1)

9000+9-1=9008 ⇒ 9008/1024=BL8
9008%1024=816 (desp2)



- 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, primerBLogico, 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 (ultimoBLogico): Calculamos 9008 / 1024 = 8 (el último bloque lógico coincide con el primero).
- Hacemos un *bread* de su bloque físico 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 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

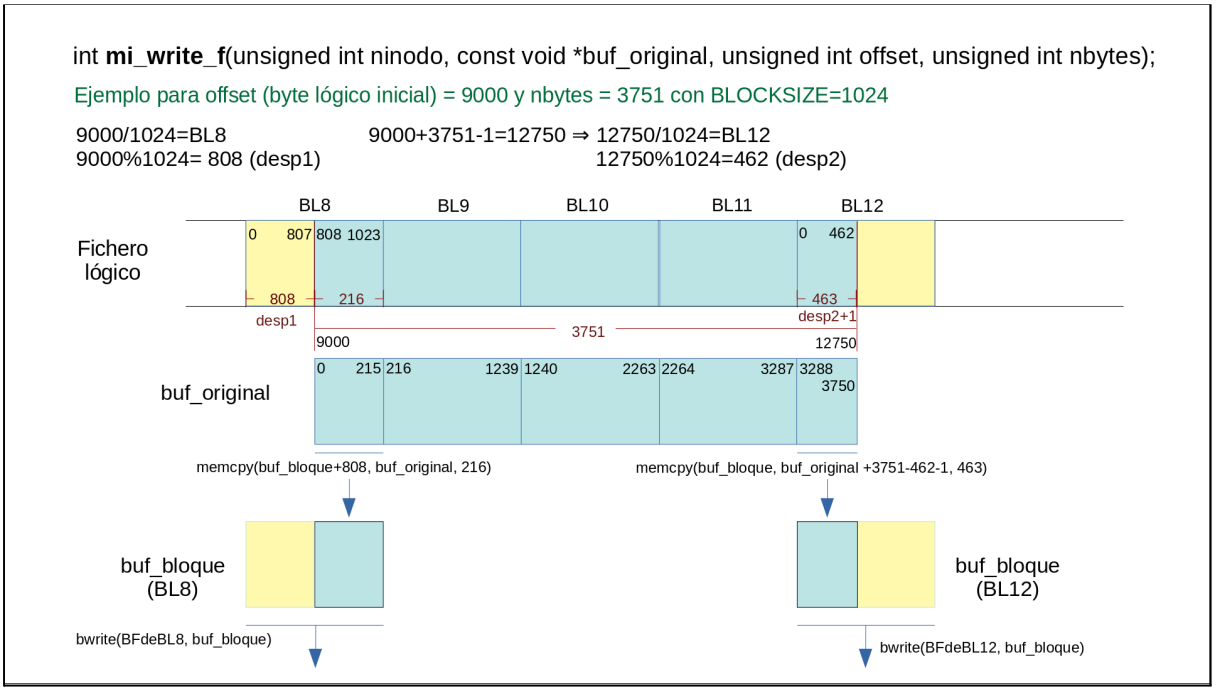
- Por tanto tendremos que copiar desp2-desp1+1 bytes (que es lo mismo que nbytes) desde el bufer_original al buf_bloque en la posición indicada por desp1:

  memcpy (buf_bloque + desp1, buf_original, desp2 +1 - desp1)

  o sea: memcpy (buf_bloque + 808, buf_original, 9)

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

Después 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 es el número 9000.
- Buscamos a qué bloque lógico pertenece, para ello calculamos primerBLogico = 9000 / 1024 = 8

- El último byte lógico, que vamos a escribir es el 12750: 9000 + 3751 – 1.
- Buscamos a qué bloque lógico pertenece: Calculamos ultimoBLogico= 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 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 paŕametro, 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 bufer_original al buf_bloque en la posición indicada por desp1:

     ```
     memcpy (buf_bloque + desp1, buf_original, BLOCKSIZE - desp1)
     ```

     o sea: 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 devuelva 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 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 - primerBLogico - 1) * BLOCKSIZE)
     ```

     y lo volcamos al dispositivo mediante un bwrite() en el bloque físico 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 buffer_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 **solo si hemos escrito más allá del final del fichero**[1]**,** y por tanto el ctime si modificamos cualquier 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*);

[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 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.

Lee información de un fichero/directorio (correspondiente al nº de inodo 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: (inodo.permisos & 4) == 4.

La función no puede leer más allá del tamaño en bytes lógicos del inodo (es decir, más allá del fin de fichero):

```
si offset >= inodo.tamEnBytesLog entonces
   leidos := 0 // No podemos leer nada
   devolver leidos
fsi
si offset + nbytes >= inodo.tamEnBytesLog entonces
   nbytes := inodo.tamEnBytesLog - offset
   // leemos sólo los bytes desde el offset hasta EOF
fsi
```

Contemplaremos los mismos casos que en escribir.c.
Hay que ir construyendo buf_original utilizando primeramente bread() para leer un bloque del dispositivo y copiando la porcion correspondiente con memcpy() al buf_original.
Tened en cuenta que las llamadas a traducir_bloque_inodo() ahora serán con **reservar=0** y que pueden devolver **-1** si el bloque físico no existe. 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.

Actualizar el atime.

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, **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**. (No es preciso utilizar campos para la alineación ni paddings)

Se recomienda definir un tipo estructurado denominado STAT (podemos considerar que la estructura STAT es la misma que la estructura INODO pero sin los punteros).

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) según indique el argumento.

Actualizar ctime!

Habrá que hacer un programita, **permitir.c,** para poder disponer desde consola de un comando que ejecute esta función (ver Anexo).

Anexo

escribir.c, leer.c y permitir.c

Son programas externos 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, 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)
- 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:

    “$(cat dir_practica/fichero.c)”

  - 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[2]
- **Tenéis que probar también con textos que ocupen más de un bloque**[3]
- 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 por pantalla (salida estándar: 1). Una vez testeado, se ha de eliminar esta llamada.
- Tenéis que probar de escribir en varios offsets para un mismo inodo
- Un ejemplo de sintaxis para esta función podría ser:

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

  Offsets: 9000, 209000, 30725000, 409605000

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

[2] Si el fichero externo ha sido generado con un editor entonces contendrá un carácter adicional
[3] Disponéis del fichero texto2.txt para las pruebas que ocupa más de 3 bloques

- 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()). Utilizar la salida estandar de errores, para ello:
  - fprintf(stderr, …)
  - o write(**2**, …), combinado con la función sprintf().

leer.c

Le pasaremos por línea de comandos un nº de inodo obtenido con el programa anterior (además del nombre del dispositivo). Su funcionamiento tiene que ser similar a la función 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): **while(mi_read_f(...)>0)**, 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 o constante simbólica para que sea fácilmente modificable.**
- El nº de bytes leídos al final tiene que ser igual al nº de bytes lógicos del fichero.
- Para mostrar los resultados por pantalla se puede utilizar el write de sistema para la salida estándard, **write(1,...)** .

  write(1, buffer_texto, leidos);[4]

- **Si justo antes de llamar a mi_read_f() inicializáis el buffer de lectura con 0s mediante memset(), 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 [5].**
- 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

[4] 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.
[5] Salvo que si el fichero externo ha sido generado con un editor entonces contendrá un carácter adicional

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

TESTS DE PRUEBA

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

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

- **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**

```
##############################################################################
$ rm disco
$ ./mi_mkfs disco 100000
#inicializamos el sistema de ficheros con 100.000 bloques
##############################################################################
$ ./leer_sf [6] 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>
```

[6] En este nivel se ha modificado leer_sf.c para que sólo muestre el superbloque. Ya no es necesario hacer llamadas externas a traducir_bloque_inodo() para ver su funcionamiento desde este programa porque serán las funciones mi_wite_f() y mi_read_f() quienes las hagan de forma interna.

```
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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

DATOS INODO 1:
tipo=f
permisos=6
atime: Tue 2020-03-17 18:56:04
ctime: Tue 2020-03-17 18:56:04
mtime: Tue 2020-03-17 18:56:04
nlinks=1
tamEnBytesLog=9009
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

DATOS INODO 1:
...
tamEnBytesLog=209009
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

DATOS INODO 1:
...
tamEnBytesLog=30725009
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

DATOS INODO 1:
```

```
...
tamEnBytesLog=409605000
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

DATOS INODO 1:
...
tamEnBytesLog=480000009
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-r--r-- 1 uib uib 480000009 mar 17 18:56 ext1.txt
################################################################################
$ ./escribir disco 123456789 1
#escribimos el texto “123456789” en los offsets 9000, 209000, 30725000,
#409605000 y 480000000, de inodos diferentes
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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

DATOS INODO 2:
...
tamEnBytesLog=9009
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

DATOS INODO 3:
...
tamEnBytesLog=209009
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

DATOS INODO 4:
...
tamEnBytesLog=30725009
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

DATOS INODO 5:
...
tamEnBytesLog=409605009
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

DATOS INODO 6:
...
tamEnBytesLog=480000009
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-r--r-- 1 uib uib 9009 mar 17 19:07 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-r--r-- 1 uib uib 409605009 mar 17 19:07 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

```
################################################################################
$ rm disco
$ ./mi_mkfs disco 100000
################################################################################
$ ./escribir disco "$(cat texto2.txt)" 1
#escribimos el texto contenido en text2.txt en los offsets  9000, 209000, 30725000,
#409605000 y 480000000 de inodos diferentes
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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)]
[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

DATOS INODO 1:
...
tamEnBytesLog=12751
```

```
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

DATOS INODO 2:
...
tamEnBytesLog=212751
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

DATOS INODO 3:
...
tamEnBytesLog=30728751
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

DATOS INODO 4:
...
tamEnBytesLog=409608751
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

DATOS INODO 5:
...
tamEnBytesLog=480003751
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-r--r-- 1 uib uib 212751 mar 18 13:54 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 (...)[7]
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

[7] Omitido para no ocupar tantas páginas. Disponéis del texto entero en texto2.txt

Nivel 6: ficheros_basico.c {liberar_inodo(), liberar_bloques_inodo()}, ficheros.c {truncar_f()}, truncar.c

En el nivel 3 ya realizamos las funciones leer_inodo(), escribir_inodo() y reservar_inodo(), ahora vamos a realizar la que nos faltaba para completar el grupo: liberar_inodo(), la cual requiere de una función auxiliar para liberar todos los bloques que cuelgan del inodo: liberar_bloques_inodo(). Con ello completaremos el conjunto de funciones de la capa ficheros_basico.c.

Después realizaremos la función truncar_f(), de la capa de ficheros que utiliza tales funciones y finalmente un programa externo, truncar.c, para llamar desde la consola a truncar_f()[1].

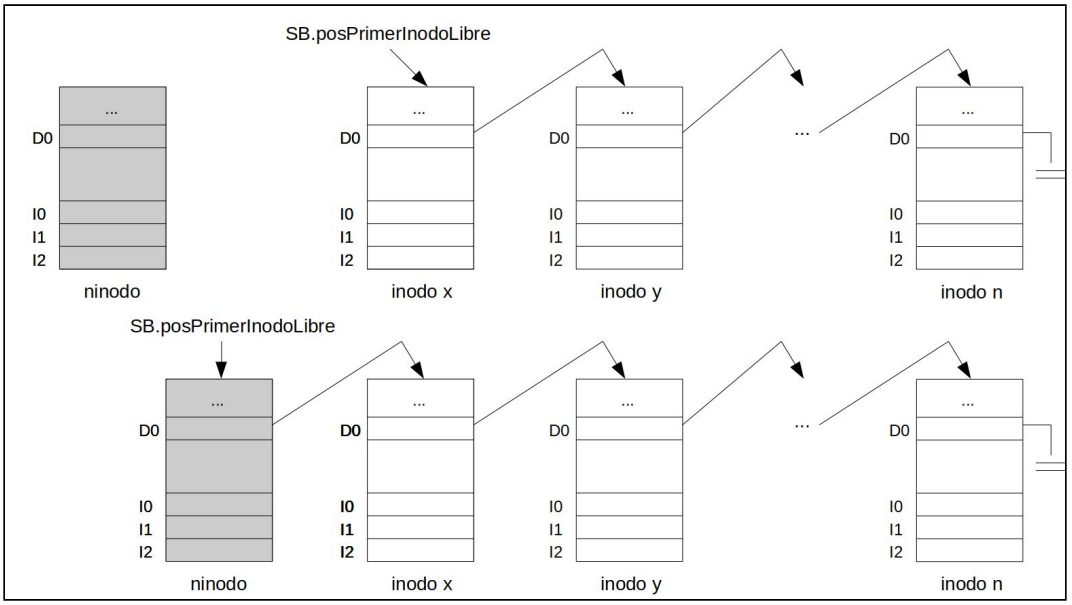
1) `int liberar_inodo(unsigned int ninodo);`

Liberar un inodo implica por un lado, que tal inodo pasará a la cabeza de la lista de inodos libres (actualizando el campo SB.posPrimerInodoLibre) y tendremos un inodo más libre en el sistema, y por otro lado, habrá que recorrer la estructura de enlaces del inodo para liberar todos aquellos bloques de datos, de la zona de datos del dispositivo virtual, que estaba ocupando, más todos aquellos bloques índice que se hubieran creado para apuntar a esos bloques.

En detalle, esta función deberá realizar las siguientes acciones:

- Leer el inodo
- Llamar a la función auxiliar liberar_bloques_inodo() para liberar todos los bloques del inodo. El argumento primerBL que le pasamos, valdrá 0 cuando la llamamos desde esta función, ya que si liberamos el inodo hemos de liberar también TODOS los bloques ocupados, desde el 0 hasta el último bloque de ese inodo (lo calcularemos a partir del tamaño en bytes lógico del inodo, leyendo ese dato en el superbloque). Hay que tener en cuenta que NO todos los **bloques lógicos** tienen porqué estar ocupados.
- A la cantidad de bloques ocupados del inodo se le restará la cantidad de bloques liberados por la función anterior (y debería quedar a 0).
- Marcar el inodo como tipo libre y tamEnBytesLog=0
- Actualizar la lista enlazada de inodos libres:
    - **Leer el superbloque** [2] para saber cuál es el primer inodo libre de la lista enlazada
    - Incluir el inodo que queremos liberar en la lista de inodos libres (por el principio), actualizando el superbloque para que éste sea ahora el primero de la lista. El inodo liberado apuntará donde antes apuntaba el campo del superbloque.

[1] truncar.c no pertenece al sistema de ficheros, es para realizar testing de las funciones mientras no tengamos completada la capa de directorios.
[2] Si lo hubiérais leído al inicio de la función de liberar_inodo() hay que tener en cuenta que liberar_bloques_inodo() cambia el valor de SB.cantBloquesLibres



Lista de inodos libres antes y después de liberar el inodo ninodo

- En el superbloque, incrementar la cantidad de **inodos libres** (la cantidad de **bloques libres** ya queda incrementada en la función liberar_bloque() cuando la llamamos desde liberar_bloque_inodo())
- Escribir el inodo
- Escribir el superbloque
- Devolver el nº del inodo liberado

Es recomendable que la parte que libera bloques se haga con la siguiente función (ya que también la usaremos dentro de la función mi_truncar_f() de **ficheros.c**):

`2) int liberar_bloques_inodo(unsigned int primerBL, struct inodo *inodo);`

La función liberar_bloques_inodo() libera todos los bloques **ocupados** (con la ayuda de la función liberar_bloque()) a partir del bloque lógico indicado por el argumento primerBL (inclusive). Esta función también la usaremos dentro de la función mi_truncar_f(), pero para liberar los bloques a partir de un determinado bloque lógico. Evidentemente, en nuestro caso desde liberar_inodo(), llamaremos a la función liberar_bloques_inodo() de manera que el argumento primerBL valga 0, dado que nos interesa liberar los bloques lógicos del inodo desde el 0, pero sólo **hasta el último bloque lógico del fichero**. Podemos calcular cuál es ese de la siguiente manera:

```
si inodo->tamEnBytesLog % BLOCKSIZE = 0 entonces
  ultimoBL := inodo->tamEnBytesLog / BLOCKSIZE - 1
si_no ultimoBL := inodo->tamEnBytesLog / BLOCKSIZE
fsi
```

Vamos liberando **los bloques que estén ocupados** con ayuda de la funcion liberar_bloque() y contabilizamos los bloques liberados. Hay que tener en cuenta que podemos tener huecos en nuestro fichero lógico, es decir, no tienen porqué estar ocupados todos los bloques lógicos anteriores ya que podemos escribir con acceso directo a cualquier posición, aunque las anteriores no estén escritas).

Hay que comprobar cuando pasemos por un bloque de punteros, si no le quedan ya punteros ocupados, puesto que **en tal caso también habría que liberar ese bloque de punteros.** Eso lo podemos averiguar utilizando la función memcmp() con un buffer auxiliar inicializado a 0s con memset()):

```
unsigned char bufAux_punteros[BLOCKSIZE]; //1024 bytes
unsigned int bloque_punteros [NPUNTEROS]; //1024 bytes
memset (bufAux_punteros, 0, BLOCKSIZE);
if (memcmp (bloque_punteros, bufAux_punteros, BLOCKSIZE)==0) ...
```

Para desarrollar liberar_bloques_inodo(), podemos optar tanto por una versión iterativa como por una versión recursiva ayudándonos, si nos son útiles, de las funciones ya implementadas obtener_nrangoBL() y obtener_indice(), y de las funciones auxiliares que consideremos necesario.

Esta función ha de devolver la **cantidad de bloques liberados**. La función que llame a ésta ya se encargará de disminuir la cantidad de bloques ocupados en el inodo en base a esa cantidad, y también de actualizar el ctime y escribir el inodo.

Ejemplo de algoritmo iterativo[3]:

```
/* Ejemplo de versión iterativa de liberar_bloques_inodo(). Hay que liberar desde el primerBL que
recibimos por parámetro hasta el último bloque lógico del inodo recorriendo todos los niveles de
punteros correspondientes */

funcion liberar_bloques_inodo(primerBL:unsigned ent, *inodo:struct inodo) devolver liberados:ent

 var
    nRangoBL, nivel_punteros, indice, ptr, nBL, ultimoBL: unsigned ent
    bloques_punteros [3] [NPUNTEROS]: ent //array de bloques de punteros
    ptr_nivel [3]: ent  //punteros a bloques de punteros de cada nivel
    indices[3]: ent  //indices de cada nivel
    liberados: ent  // nº de bloques liberados
 fvar

 liberados:=0
 si inodo->tamEnBytesLog = 0 entonces devolver 0 fsi // el fichero está vacío
 //obtenemos el último bloque lógico del inodo
```

[3] Este algoritmo es muy ineficiente, habría que minimizar los accesos a disco (funciones bwrite() y bread() y también saltar la exploración de los bloques logicos de cualquier ramificación cuando la raíz de ésta es igual a 0

```
si inodo->tamEnBytesLog % BLOCKSIZE = 0 entonces
   ultimoBL := inodo->tamEnBytesLog / BLOCKSIZE - 1
si_no  ultimoBL := inodo->tamEnBytesLog / BLOCKSIZE
fsi
ptr:= 0
para nBL := primerBL hasta nBL = ultimoBL paso 1 hacer //recorrido BLs [4]
   nRangoBL := obtener_nrangoBL(*inodo, nBL, &ptr) //0:D, 1:I0, 2:I1, 3:I2
   si nRangoBL < 0 entonces devolver ERROR fsi
   nivel_punteros := nRangoBL //el nivel_punteros +alto cuelga del inodo

   mientras (ptr > 0 && nivel_punteros > 0) hacer //cuelgan bloques de punteros
      indice := obtener_indice(nBL, nivel_punteros)
      si indice=0  o nBL=primerBL entonces
        //solo leemos del dispositivo si no está ya cargado en un buffer
         bread(ptr, bloques_punteros[nivel_punteros - 1]
      fsi
      ptr_nivel[nivel_punteros-1] := ptr
      indices[nivel_punteros-1] := indice
      ptr := bloques_punteros[nivel_punteros-1][indice]
      nivel_punteros--
   fmientras

   si ptr > 0 entonces //si existe bloque de datos
      liberar_bloque(ptr)
      liberados++
      si nRangoBL = 0 entonces //es un puntero Directo
        inodo->punterosDirectos[nBL]:= 0
      si_no
         mientras nivel_punteros < nRangoBL hacer
            indice := indices[nivel_punteros]
            bloques_punteros[nivel_punteros][indice] := 0
            ptr := ptr_nivel [nivel_punteros]
            si bloques_punteros[nivel_punteros] = 0 [5] entonces
               //No cuelgan bloques ocupados, hay que liberar el bloque de punteros
               liberar_bloque(ptr)
               liberados++
               nivel_punteros++
               si nivel_punteros = nRangoBL entonces
                  inodo->punterosIndirectos[nRangoBL-1] := 0
               fsi
            si_no  //escribimos en el dispositivo el bloque de punteros modificado
               bwrite(ptr, bloques_punteros[nivel_punteros]) [6]
               nivel_punteros := nRangoBL
               // para salir del bucle ya que no habrá que liberar los bloques de niveles
```

[4] No es necesario recorrerlos todos. Cuando encontremos un puntero=0, tanto en el inodo como en algún nivel de punteros, podemos descartar de golpe todos los BLs que colgarían de él.
[5] Utilizar la función memcmp() y un buffer auxiliar de unsigned char de tamaño BLOCKSIZE inicializado a 0, como se ha explicado anteriormente
[6] Se puede mejorar escribiendo en el dispositivo solamente cuando hemos acabado de explorar ese bloque en memoria

```
                //superiores de los que cuelga
            fsi
        fmientras
    fsi
  fsi
 fpara
 devolver liberados
ffuncion
```

Hay que incluir la declaración de las funciones anteriores en el fichero ficheros_basico.h

A continuación viene una función de la capa de ficheros que también hace uso de liberar_bloques_inodo() pero liberando desde un determinado BL hasta el final:

3) int mi_truncar_f(unsigned int n*inodo*, unsigned int *nbytes*);

Trunca un fichero/directorio (correspondiente al nº de inodo pasado como argumento) a los bytes indicados, liberando los bloques necesarios.
En nuestro sistema de ficheros, esta función será llamada desde la función mi_unlink() de la capa de directorios, la cuál a su vez será llamada desde el programa mi_rm.c, y nos servirá para eliminar una entrada de un directorio.

Hay que comprobar que el inodo tenga permisos de escritura.
No se puede truncar más allá del tamaño en bytes lógicos del fichero/directorio.

Nos basaremos en la función liberar_bloques_inodo() . Para saber que nº de bloque lógico le hemos de pasar como primer bloque lógico a liberar:

```
si nbytes % BLOCKSIZE = 0  entonces primerBL := nbytes/BLOCKSIZE
si_no primerBL := nbytes/BLOCKSIZE + 1
```

Actualizar mtime, ctime, el tamaño en bytes lógicos del inodo (**pasará a ser igual a nbytes**) y el número de bloques ocupados del inodo (habrá que restarle los liberados).

Devolver la cantidad de bloques liberados.

A continuación habrá que hacer un programita, **truncar.c**, para poder disponer desde consola de un comando que ejecute esta función:

truncar.c

Sintaxis: truncar <nombre_dispositivo> <ninodo> <nbytes>
Pasos a realizar:

- Validación de sintaxis
- montar dispositivo virtual
- si nbytes= 0 liberar_inodo() si no mi_truncar_f() fsi
- desmontar dispositivo virtual

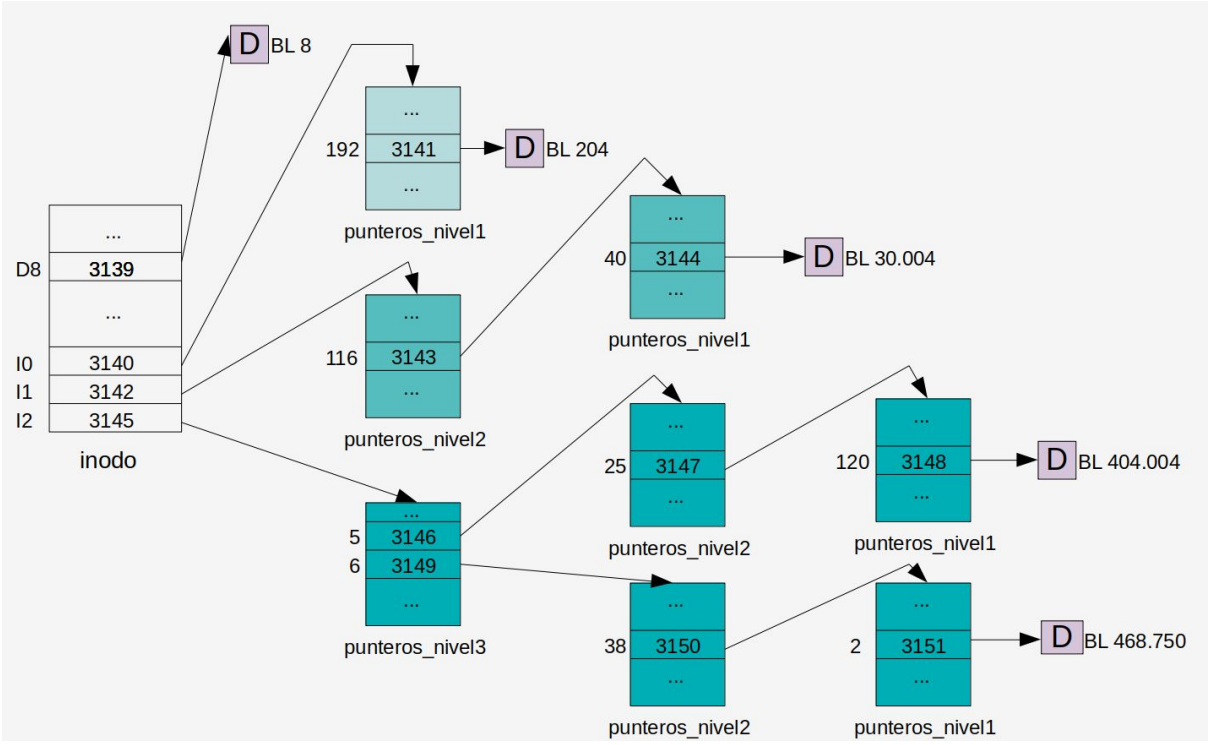
nbytes puede ser 0 (simularía la llamada de mi_truncar_f() desde liberar_inodo()), o bien un valor que al menos deje 1 byte escrito en el inodo ya que si no el campo tamEnbytesLog se quedará incoherente. Cuando esté la práctica completa nunca indicaremos nbytes desde consola, es sólo ahora para poder testear nuestras funciones.
No sólo hay que comprobar que el tamaño del inodo pasa a ser nbytes sino también que se libera la cantidad de bloques correcta (han de quedar con valor correcto los campos numBloquesOcupados del inodo y cantBloquesLibres del superbloque):

- La **cantBloquesLibres** del SB sólo la debería decrementar la función **reservar_bloque()**, y solo lo debería incrementar la función **liberar_bloques()**.
- El **numBloquesOcupados** del inodo sólo lo debería incrementar la función **traducir_bloque_inodo()** con reservar=1, y sólo lo debería decrementar la función **mi_truncar_f()** o **liberar_inodo()**.

Al finalizar llamad a mi_stat_f() para mostrar al menos el tamEnBytesLog y numBloquesOcupados para comprobar que son correctos.

TESTS DE PRUEBA

- Ejecución de **script_truncar.sh** que escribe “123456789” en diferentes offsets de un mismo inodo (9.000 ⊂ BL 8, 209.000 ⊂ BL 204, 30.725.000 ⊂ BL 30.004, 409.605.000 ⊂ BL 404.004, 480.000.000 ⊂ BL 468.750) y luego elimina el inodo con todos sus bloques



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

$ ./mi_mkfs disco 100000
$ ./leer_sf disco

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 disco "123456789" 0
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=9009
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=209009
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=30725009
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=409605009
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=480000009
numBloquesOcupados=13
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96848
cantInodosLibres = 24998
...

$ ./truncar disco 1 0
[liberar_bloques_inodo()→ primer BL: 0, último BL: 468750]
[liberar_bloques_inodo()→ liberamos BF 3139: datos de BL 8]
[liberar_bloques_inodo()→ liberamos BF 3141: datos de BL 204]
[liberar_bloques_inodo()→ liberamos BF 3140: bloque de punteros_nivel1 de indirectos0]
[liberar_bloques_inodo()→ liberamos BF 3144: datos de BL 30004]
[liberar_bloques_inodo()→ liberamos BF 3143: bloque de punteros_nivel1 de indiretos1]
[liberar_bloques_inodo()→ liberamos BF 3142: bloque de punteros_nivel2 de indiretos1]
[liberar_bloques_inodo()→ liberamos BF 3148: datos de BL 400004]
[liberar_bloques_inodo()→ liberamos BF 3147: bloque de punteros_nivel1 de indiretos2]
[liberar_bloques_inodo()→ liberamos BF 3146: bloque de punteros_nivel2 de indiretos2]
[liberar_bloques_inodo()→ liberamos BF 3151: datos de BL 468750]
[liberar_bloques_inodo()→ liberamos BF 3150: bloque de punteros_nivel1 de indiretos2]
[liberar_bloques_inodo()→ liberamos BF 3149: bloque de punteros_nivel2 de indiretos2]
[liberar_bloques_inodo()→ liberamos BF 3145: bloque de punteros_nivel3 de indiretos2]
[liberar_bloques_inodo()→ total bloques liberados: 13]
```

```
[liberar_inodo() → liberados: 13]

DATOS INODO 1:
tipo=l
...
tamEnBytesLog=0
numBloquesOcupados=0
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 1
cantBloquesLibres = 96861
cantInodosLibres = 24999
...
```

- Ejecución de **script_truncar_parcial.sh** que escribe “123456789” en diferentes offsets de un mismo inodo y luego va haciendo truncamientos sucesivos desde diferentes bytes, que van recortando el tamaño en bytes lógico del fichero.

```
$ ./mi_mkfs disco 100000
$ ./escribir disco "123456789" 0
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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

DATOS INODO 1:
tipo=f
permisos=6
...
tamEnBytesLog=9009
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

DATOS INODO 1:
tipo=f
```

```
...
tamEnBytesLog=209009
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=30725009
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=409605009
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

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=480000009
numBloquesOcupados=13
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
```

```
cantBloquesLibres = 96848
cantInodosLibres = 24998
...

$ ./truncar disco 1 409605001[7]
[liberar_bloques_inodo()→ primer BL: 400005, último BL: 468750]
[liberar_bloques_inodo()→ liberado BF 3151 de datos para BL 468750]
[liberar_bloques_inodo()→ liberado BF 3150 de punteros_nivel1 correspondiente al BL 468750]
[liberar_bloques_inodo()→ liberado BF 3149 de punteros_nivel2 correspondiente al BL 468750]
[liberar_bloques_inodo()→ total bloques liberados: 3]

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=409605001
numBloquesOcupados=10
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96851
cantInodosLibres = 24998
...

$ ./truncar disco 1 30725003[8]
[liberar_bloques_inodo()→ primer BL: 30005, último BL: 400004]
[liberar_bloques_inodo()→ liberamos BF 3148: datos de BL 400004]
[liberar_bloques_inodo()→ liberado BF 3147 de punteros_nivel1 correspondiente al BL 400004]
[liberar_bloques_inodo()→ liberado BF 3146 de punteros_nivel2 correspondiente al BL 400004]
[liberar_bloques_inodo()→ liberado BF 3145 de punteros_nivel3 correspondiente al BL 400004]
[liberar_bloques_inodo()→ total bloques liberados: 4]

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=30725003
numBloquesOcupados=6
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96855
cantInodosLibres = 24998
...
```

[7] byte 409.605.001 ⊂ BL 400.004 (hay que preservarlo ya que aún contiene otros bytes y empezar a liberar a partir del siguiente)

[8] byte 30.725.003 ⊂ BL 30.004 (hay que preservarlo ya que aún contiene otros bytes y empezar a liberar a partir del siguiente)

$ ./truncar disco 1 209008[9]
[liberar_bloques_inodo()→ primer BL: 205, último BL: 30004]
[liberar_bloques_inodo()→ liberamos BF 3144: datos de BL 30004]
[liberar_bloques_inodo()→ liberado BF 3143 de punteros_nivel1 correspondiente al BL 30004]
[liberar_bloques_inodo()→ liberado BF 3142 de punteros_nivel2 correspondiente al BL 30004]
[liberar_bloques_inodo()→ total bloques liberados: 3]

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=209008
numBloquesOcupados=3
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96858
cantInodosLibres = 24998
...

$ ./truncar disco 1 9005[10]
[liberar_bloques_inodo()→ primer BL: 9, último BL: 204]
[liberar_bloques_inodo()→ liberamos BF 3141: datos de BL 204]
[liberar_bloques_inodo()→ liberado BF 3140 de punteros_nivel1 correspondiente al BL 204]
[liberar_bloques_inodo()→ total bloques liberados: 2]

DATOS INODO 1:
tipo=f
...
tamEnBytesLog=9005
numBloquesOcupados=1
$ ./leer_sf disco

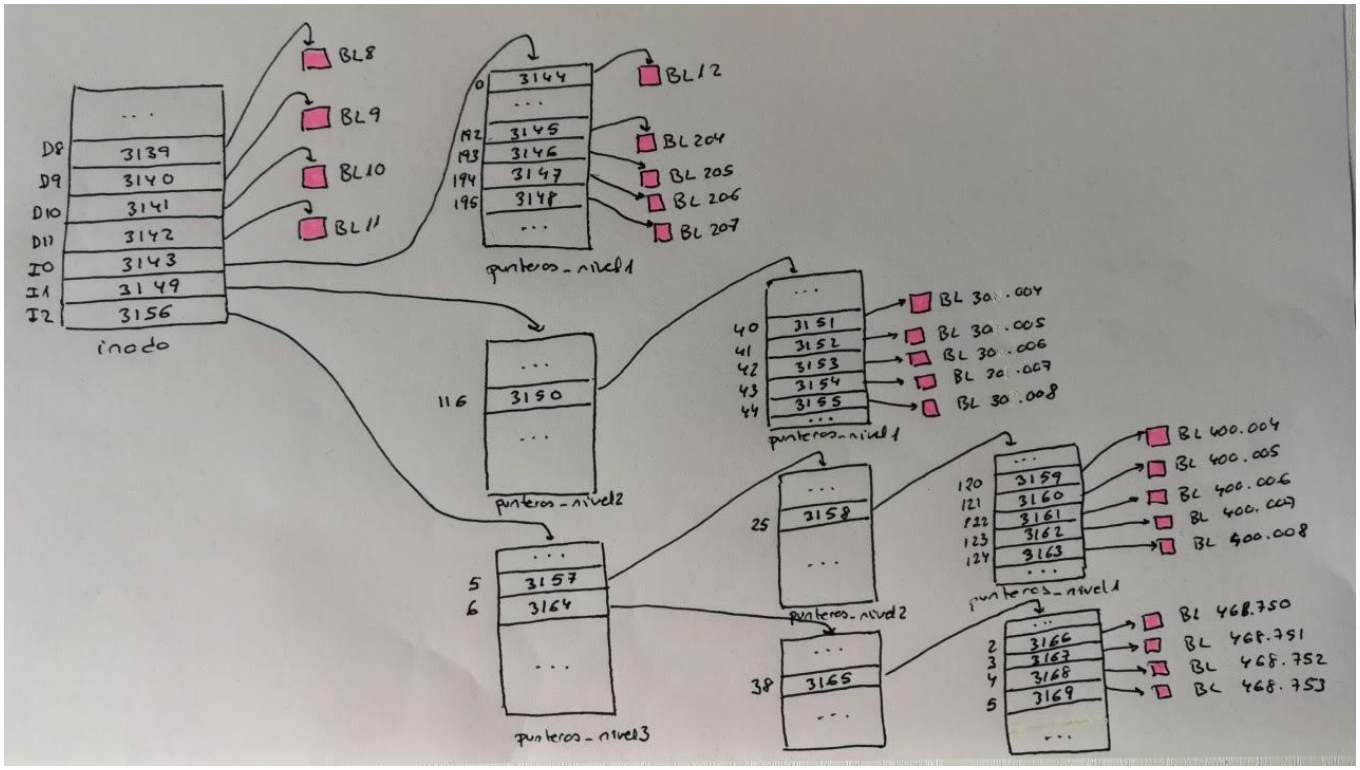
DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96860
cantInodosLibres = 24998
...

$ ./truncar disco 1 0
[liberar_bloques_inodo()→ primer BL: 0, último BL: 8]
[liberar_bloques_inodo()→ liberamos BF 3139 de datos: BL 8]
[liberar_bloques_inodo()→ total bloques liberados: 1]

[9] byte 209.008 ⊂ BL 204 (hay que preservarlo ya que aún contiene otros bytes y empezar a liberar a partir del siguiente)
[10] byte 9.005 ⊂ BL 8 (hay que preservarlo ya que aún contiene otros bytes y empezar a liberar a partir del siguiente)

```
[liberar_inodo() → liberados: 1]

DATOS INODO 1:
tipo=l
...
tamEnBytesLog=0
numBloquesOcupados=0
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 1
cantBloquesLibres = 96861
cantInodosLibres = 24999
...
```

- Ejecución de **script_truncar2.sh** que escribe el texto2.txt, que ocupa varios bloques, en diferentes offsets de un mismo inodo y luego libera el inodo y todos sus bloques




```
$ ./mi_mkfs disco 100000
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 1
```

```
cantBloquesLibres = 96861
cantInodosLibres = 24999
...

$ ./escribir disco "$(cat texto2.txt)" 0
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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)]
[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

DATOS INODO 1:
...
tamEnBytesLog=12751
numBloquesOcupados=6

Nº inodo reservado: 1
offset: 209000
[traducir_bloque_inodo()→ punteros_nivel1 [192] = 3145 (reservado BF 3145 para BL 204)]
[traducir_bloque_inodo()→ punteros_nivel1 [193] = 3146 (reservado BF 3146 para BL 205)]
[traducir_bloque_inodo()→ punteros_nivel1 [194] = 3147 (reservado BF 3147 para BL 206)]
[traducir_bloque_inodo()→ punteros_nivel1 [195] = 3148 (reservado BF 3148 para BL 207)]
Bytes escritos: 3751

DATOS INODO 1:
...
tamEnBytesLog=212751
numBloquesOcupados=10

Nº inodo reservado: 1
offset: 30725000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[1] = 3149 (reservado BF 3149 para
punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [116] = 3150 (reservado BF 3150 para
punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [40] = 3151 (reservado BF 3151 para BL 30004)]
[traducir_bloque_inodo()→ punteros_nivel1 [41] = 3152 (reservado BF 3152 para BL 30005)]
[traducir_bloque_inodo()→ punteros_nivel1 [42] = 3153 (reservado BF 3153 para BL 30006)]
[traducir_bloque_inodo()→ punteros_nivel1 [43] = 3154 (reservado BF 3154 para BL 30007)]
```

```
[traducir_bloque_inodo()→ punteros_nivel1 [44] = 3155 (reservado BF 3155 para BL 30008)]
Bytes escritos: 3751

DATOS INODO 1:
...
tamEnBytesLog=30728751
numBloquesOcupados=17

Nº inodo reservado: 1
offset: 409605000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3156 (reservado BF 3156 para
punteros_nivel3)]
[traducir_bloque_inodo()→ punteros_nivel3 [5] = 3157 (reservado BF 3157 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [25] = 3158 (reservado BF 3158 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [120] = 3159 (reservado BF 3159 para BL 400004)]
[traducir_bloque_inodo()→ punteros_nivel1 [121] = 3160 (reservado BF 3160 para BL 400005)]
[traducir_bloque_inodo()→ punteros_nivel1 [122] = 3161 (reservado BF 3161 para BL 400006)]
[traducir_bloque_inodo()→ punteros_nivel1 [123] = 3162 (reservado BF 3162 para BL 400007)]
[traducir_bloque_inodo()→ punteros_nivel1 [124] = 3163 (reservado BF 3163 para BL 400008)]
Bytes escritos: 3751

DATOS INODO 1:
...
tamEnBytesLog=409608751
numBloquesOcupados=25

Nº inodo reservado: 1
offset: 480000000
[traducir_bloque_inodo()→ punteros_nivel3 [6] = 3164 (reservado BF 3164 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [38] = 3165 (reservado BF 3165 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [2] = 3166 (reservado BF 3166 para BL 468750)]
[traducir_bloque_inodo()→ punteros_nivel1 [3] = 3167 (reservado BF 3167 para BL 468751)]
[traducir_bloque_inodo()→ punteros_nivel1 [4] = 3168 (reservado BF 3168 para BL 468752)]
[traducir_bloque_inodo()→ punteros_nivel1 [5] = 3169 (reservado BF 3169 para BL 468753)]
Bytes escritos: 3751

DATOS INODO 1:
...
tamEnBytesLog=480003751
numBloquesOcupados=31

$ ./truncar disco 1 0
[liberar_bloques_inodo()→ primer BL: 0, último BL: 468753]
[liberar_bloques_inodo()→ liberado BF 3139 de datos para BL 8]
[liberar_bloques_inodo()→ liberado BF 3140 de datos para BL 9]
[liberar_bloques_inodo()→ liberado BF 3141 de datos para BL 10]
[liberar_bloques_inodo()→ liberado BF 3142 de datos para BL 11]
```

```
[liberar_bloques_inodo()→ liberado BF 3144 de datos para BL 12]
[liberar_bloques_inodo()→ liberado BF 3145 de datos para BL 204]
[liberar_bloques_inodo()→ liberado BF 3146 de datos para BL 205]
[liberar_bloques_inodo()→ liberado BF 3147 de datos para BL 206]
[liberar_bloques_inodo()→ liberado BF 3148 de datos para BL 207]
[liberar_bloques_inodo()→ liberado BF 3143 de punteros_nivel1 correspondiente al BL 207]
[liberar_bloques_inodo()→ liberado BF 3151 de datos para BL 30004]
[liberar_bloques_inodo()→ liberado BF 3152 de datos para BL 30005]
[liberar_bloques_inodo()→ liberado BF 3153 de datos para BL 30006]
[liberar_bloques_inodo()→ liberado BF 3154 de datos para BL 30007]
[liberar_bloques_inodo()→ liberado BF 3155 de datos para BL 30008]
[liberar_bloques_inodo()→ liberado BF 3150 de punteros_nivel1 correspondiente al BL 30008]
[liberar_bloques_inodo()→ liberado BF 3149 de punteros_nivel2 correspondiente al BL 30008]
[liberar_bloques_inodo()→ liberado BF 3159 de datos para BL 400004]
[liberar_bloques_inodo()→ liberado BF 3160 de datos para BL 400005]
[liberar_bloques_inodo()→ liberado BF 3161 de datos para BL 400006]
[liberar_bloques_inodo()→ liberado BF 3162 de datos para BL 400007]
[liberar_bloques_inodo()→ liberado BF 3163 de datos para BL 400008]
[liberar_bloques_inodo()→ liberado BF 3158 de punteros_nivel1 correspondiente al BL 400008]
[liberar_bloques_inodo()→ liberado BF 3157 de punteros_nivel2 correspondiente al BL 400008]
[liberar_bloques_inodo()→ liberado BF 3166 de datos para BL 468750]
[liberar_bloques_inodo()→ liberado BF 3167 de datos para BL 468751]
[liberar_bloques_inodo()→ liberado BF 3168 de datos para BL 468752]
[liberar_bloques_inodo()→ liberado BF 3169 de datos para BL 468753]
[liberar_bloques_inodo()→ liberado BF 3165 de punteros_nivel1 correspondiente al BL 468753]
[liberar_bloques_inodo()→ liberado BF 3164 de punteros_nivel2 correspondiente al BL 468753]
[liberar_bloques_inodo()→ liberado BF 3156 de punteros_nivel3 correspondiente al BL 468753]
[liberar_bloques_inodo()→ total bloques liberados: 31]

DATOS INODO 1:
tipo=l
...
tamEnBytesLog=0
numBloquesOcupados=0
```

- Ejecución de **script_truncar_parcial2.sh** que escribe el texto2.txt, que ocupa varios bloques, en diferentes offsets de un mismo inodo y luego va haciendo truncamientos sucesivos que van recortando el tamaño en bytes lógico del fichero.
    - El primer truncamiento es en el byte 409.605.001 ⊂ BL 400.004, y quedan más bytes escritos en ese BL, por tanto no se puede eliminar y supondrá liberar los BL ocupados desde el 400.005 al 468.753
    - El segundo truncamiento es en el byte 30.725.003 ⊂ BL 30.004, y quedan más bytes escritos en ese BL, por tanto no se puede eliminar y supondrá liberar los BL ocupados desde el 30.005 al 400.004

- El tercer truncamiento es en el byte 209.008 ⊂ BL 204, y quedan más bytes escritos en ese BL, por tanto no se puede eliminar y supondrá liberar los BL ocupados desde el 205 al 30.004
- El cuarto truncamiento es en el byte 9.005 ⊂ BL 8, y quedan más bytes escritos en ese BL, por tanto no se puede eliminar y supondrá liberar los BL ocupados desde el 9 al 204
- El último truncamiento es en el byte 0 y eso implica liberar el inodo y los bloques ocupados restantes (BL 8)

```
$ ./mi_mkfs disco 100000
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 1
cantBloquesLibres = 96861
cantInodosLibres = 24999
...

$ ./escribir disco  "$(cat texto2.txt)" 0
Offsets: 9.000, 209.000, 30.725.000, 409.605.000, 480.000.000
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)]
[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

DATOS INODO 1:
...
tamEnBytesLog=12751
numBloquesOcupados=6

Nº inodo reservado: 1
offset: 209000
[traducir_bloque_inodo()→ punteros_nivel1 [192] = 3145 (reservado BF 3145 para BL 204)]
[traducir_bloque_inodo()→ punteros_nivel1 [193] = 3146 (reservado BF 3146 para BL 205)]
[traducir_bloque_inodo()→ punteros_nivel1 [194] = 3147 (reservado BF 3147 para BL 206)]
[traducir_bloque_inodo()→ punteros_nivel1 [195] = 3148 (reservado BF 3148 para BL 207)]
Bytes escritos: 3751

DATOS INODO 1:
```

```
...
tamEnBytesLog=212751
numBloquesOcupados=10

Nº inodo reservado: 1
offset: 30725000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[1] = 3149 (reservado BF 3149 para
punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [116] = 3150 (reservado BF 3150 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [40] = 3151 (reservado BF 3151 para BL 30004)]
[traducir_bloque_inodo()→ punteros_nivel1 [41] = 3152 (reservado BF 3152 para BL 30005)]
[traducir_bloque_inodo()→ punteros_nivel1 [42] = 3153 (reservado BF 3153 para BL 30006)]
[traducir_bloque_inodo()→ punteros_nivel1 [43] = 3154 (reservado BF 3154 para BL 30007)]
[traducir_bloque_inodo()→ punteros_nivel1 [44] = 3155 (reservado BF 3155 para BL 30008)]
Bytes escritos: 3751

DATOS INODO 1:
...
tamEnBytesLog=30728751
numBloquesOcupados=17

Nº inodo reservado: 1
offset: 409605000
[traducir_bloque_inodo()→ inodo.punterosIndirectos[2] = 3156 (reservado BF 3156 para
punteros_nivel3)]
[traducir_bloque_inodo()→ punteros_nivel3 [5] = 3157 (reservado BF 3157 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [25] = 3158 (reservado BF 3158 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [120] = 3159 (reservado BF 3159 para BL 400004)]
[traducir_bloque_inodo()→ punteros_nivel1 [121] = 3160 (reservado BF 3160 para BL 400005)]
[traducir_bloque_inodo()→ punteros_nivel1 [122] = 3161 (reservado BF 3161 para BL 400006)]
[traducir_bloque_inodo()→ punteros_nivel1 [123] = 3162 (reservado BF 3162 para BL 400007)]
[traducir_bloque_inodo()→ punteros_nivel1 [124] = 3163 (reservado BF 3163 para BL 400008)]
Bytes escritos: 3751

DATOS INODO 1:
...
tamEnBytesLog=409608751
numBloquesOcupados=25

Nº inodo reservado: 1
offset: 480000000
[traducir_bloque_inodo()→ punteros_nivel3 [6] = 3164 (reservado BF 3164 para punteros_nivel2)]
[traducir_bloque_inodo()→ punteros_nivel2 [38] = 3165 (reservado BF 3165 para punteros_nivel1)]
[traducir_bloque_inodo()→ punteros_nivel1 [2] = 3166 (reservado BF 3166 para BL 468750)]
[traducir_bloque_inodo()→ punteros_nivel1 [3] = 3167 (reservado BF 3167 para BL 468751)]
[traducir_bloque_inodo()→ punteros_nivel1 [4] = 3168 (reservado BF 3168 para BL 468752)]
[traducir_bloque_inodo()→ punteros_nivel1 [5] = 3169 (reservado BF 3169 para BL 468753)]
```

```
Bytes escritos: 3751

DATOS INODO 1:
...
tamEnBytesLog=480003751
numBloquesOcupados=31
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96830
cantInodosLibres = 24998
...

$ ./truncar disco 1 409605001
[liberar_bloques_inodo()→ primer BL: 400005, último BL: 468753]
[liberar_bloques_inodo()→ liberado BF 3160 de datos para BL 400005]
[liberar_bloques_inodo()→ liberado BF 3161 de datos para BL 400006]
[liberar_bloques_inodo()→ liberado BF 3162 de datos para BL 400007]
[liberar_bloques_inodo()→ liberado BF 3163 de datos para BL 400008]
[liberar_bloques_inodo()→ liberado BF 3166 de datos para BL 468750]
[liberar_bloques_inodo()→ liberado BF 3167 de datos para BL 468751]
[liberar_bloques_inodo()→ liberado BF 3168 de datos para BL 468752]
[liberar_bloques_inodo()→ liberado BF 3169 de datos para BL 468753]
[liberar_bloques_inodo()→ liberado BF 3165 de punteros_nivel1 correspondiente al BL 468753]
[liberar_bloques_inodo()→ liberado BF 3164 de punteros_nivel2 correspondiente al BL 468753]
[liberar_bloques_inodo()→ total bloques liberados: 10]

DATOS INODO 1:
tipo=f
permisos=6
atime: Fri 2020-03-27 19:37:53
ctime: Fri 2020-03-27 19:37:53
mtime: Fri 2020-03-27 19:37:53
nlinks=1
tamEnBytesLog=409605001
numBloquesOcupados=21
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96840
cantInodosLibres = 24998
...
```

$ ./truncar disco 1 30725003
[liberar_bloques_inodo()→ primer BL: 30005, último BL: 400004]
[liberar_bloques_inodo()→ liberado BF 3152 de datos para BL 30005]
[liberar_bloques_inodo()→ liberado BF 3153 de datos para BL 30006]
[liberar_bloques_inodo()→ liberado BF 3154 de datos para BL 30007]
[liberar_bloques_inodo()→ liberado BF 3155 de datos para BL 30008]
[liberar_bloques_inodo()→ liberado BF 3159 de datos para BL 400004]
[liberar_bloques_inodo()→ liberado BF 3158 de punteros_nivel1 correspondiente al BL 400004]
[liberar_bloques_inodo()→ liberado BF 3157 de punteros_nivel2 correspondiente al BL 400004]
[liberar_bloques_inodo()→ liberado BF 3156 de punteros_nivel3 correspondiente al BL 400004]
[liberar_bloques_inodo()→ total bloques liberados: 8]

DATOS INODO 1:
tipo=f
permisos=6
atime: Fri 2020-03-27 19:37:53
ctime: Fri 2020-03-27 19:37:53
mtime: Fri 2020-03-27 19:37:53
nlinks=1
tamEnBytesLog=30725003
numBloquesOcupados=13
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96848
cantInodosLibres = 24998
...

$ ./truncar disco 1 209008
[liberar_bloques_inodo()→ primer BL: 205, último BL: 30004]
[liberar_bloques_inodo()→ liberado BF 3146 de datos para BL 205]
[liberar_bloques_inodo()→ liberado BF 3147 de datos para BL 206]
[liberar_bloques_inodo()→ liberado BF 3148 de datos para BL 207]
[liberar_bloques_inodo()→ liberado BF 3151 de datos para BL 30004]
[liberar_bloques_inodo()→ liberado BF 3150 de punteros_nivel1 correspondiente al BL 30004]
[liberar_bloques_inodo()→ liberado BF 3149 de punteros_nivel2 correspondiente al BL 30004]
[liberar_bloques_inodo()→ total bloques liberados: 6]

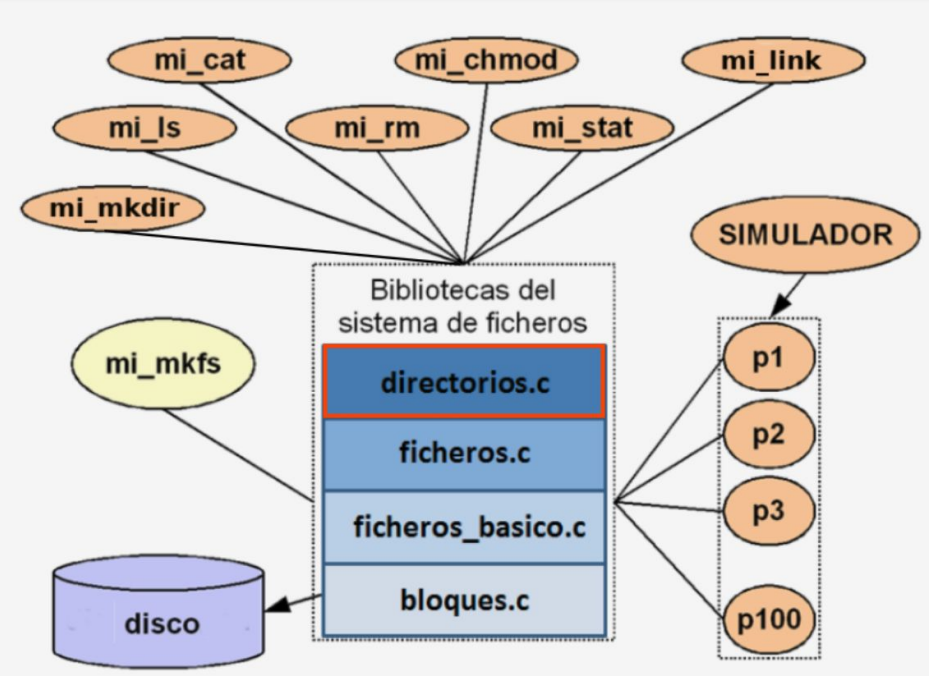
DATOS INODO 1:
tipo=f
permisos=6
atime: Fri 2020-03-27 19:37:53
ctime: Fri 2020-03-27 19:37:53
mtime: Fri 2020-03-27 19:37:53
nlinks=1

```
tamEnBytesLog=209008
numBloquesOcupados=7
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96854
cantInodosLibres = 24998
...

$ ./truncar disco 1 9005
[liberar_bloques_inodo()→ primer BL: 9, último BL: 204]
[liberar_bloques_inodo()→ liberado BF 3140 de datos para BL 9]
[liberar_bloques_inodo()→ liberado BF 3141 de datos para BL 10]
[liberar_bloques_inodo()→ liberado BF 3142 de datos para BL 11]
[liberar_bloques_inodo()→ liberado BF 3144 de datos para BL 12]
[liberar_bloques_inodo()→ liberado BF 3145 de datos para BL 204]
[liberar_bloques_inodo()→ liberado BF 3143 de punteros_nivel1 correspondiente al BL 204]
[liberar_bloques_inodo()→ total bloques liberados: 6]

DATOS INODO 1:
tipo=f
permisos=6
atime: Fri 2020-03-27 19:37:53
ctime: Fri 2020-03-27 19:37:53
mtime: Fri 2020-03-27 19:37:53
nlinks=1
tamEnBytesLog=9005
numBloquesOcupados=1
$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 2
cantBloquesLibres = 96860
cantInodosLibres = 24998
...

$ ./truncar disco 1 0
[liberar_bloques_inodo()→ primer BL: 0, último BL: 8]
[liberar_bloques_inodo()→ liberado BF 3139 de datos para BL 8]
[liberar_bloques_inodo()→ total bloques liberados: 1]

DATOS INODO 1:
tipo=l
...
```

```
tamEnBytesLog=0
numBloquesOcupados=0
$ ./leer_sf disco
DATOS DEL SUPERBLOQUE
...
posPrimerInodoLibre = 1
cantBloquesLibres = 96861
cantInodosLibres = 24999
...
```

Nivel 7: directorios.c {extraer_camino(), buscar_entrada()}

Ahora que ya tenemos las funciones de las 3 capas inferiores de nuestro sistema de ficheros, comenzaremos con el desarrollo de la última capa **directorios.c**, y los programas externos (comandos) que permitirán operar con nuestro sistema.



Un directorio no es más que un fichero especial (inodo tipo=d) en el que se van almacenando entradas: cada entrada hace referencia a un directorio o fichero contenido dentro de él.

La estructura que se usará para cada entrada (definida en directorios.h)[1] estará formada por los siguientes campos:

Nombre de archivo	ninodo

```
struct entrada {
        char nombre[60];  //En el SF ext2 la longitud del nombre es 256
        unsigned int ninodo;
};
```

Este campo *nombre* **no incluye el camino de directorios** (ni el carácter de separación '/').

[1] Hay que incluir directorios.h en todos los programas.c que hagamos a partir de ahora y, también en mi_mkfs.c y leer_sf.c, en vez de ficheros.h.

Con esta estructura, en un bloque de 1024 bytes cabrían 16 entradas de directorio.

A partir de este momento, los directorios y los ficheros serán conocidos por su ruta+nombre (en vez de por el número de su inodo).
La ruta es el camino de directorios seguido para llegar a él. Para facilitar el trabajo, haremos que el nombre de los directorios termine con el carácter de separación '/' (el de de los ficheros no terminará con ningún carácter en particular).

- Ejemplo de ruta+nombre de directorio: "/dir1/dir2/dir3/"
- Ejemplo de ruta+nombre de fichero: "/dir1/dir2/fich"

Cuando se crea un fichero, no sólo se le tiene que asignar un inodo sino que también hay que crear una entrada de directorio. Un **fichero huérfano** es aquél que no tiene entrada de directorio.
Puede haber varios nombres de archivos, distribuídos por la jerarquía de directorios, que estén ligados a un mismo inodo (**enlaces**), en tal caso se requerirá hacer uso de un contador de enlaces (nlinks). Cuando se elimine una entrada de directorio, se decrementará la cantidad de enlaces pero no se podrá eliminar el inodo correspondiente si todavía quedan enlaces.

La localización de una entrada de directorio se realiza con una búsqueda lineal entre todas las entradas de ese directorio (es muy costoso). Añadir una entrada supondrá buscarla y si no existe, la añadiremos siempre por el final.

Una primera función que podemos definir es la siguiente:

1) int extraer_camino(const char *_camino_, char *_inicial_, char *_final, char *tipo_);

Dada una cadena de caracteres camino *(*que comience por '/'), separa su contenido en dos:

- Guarda en *inicial la porción de *camino comprendida entre los dos primeros '/' (entonces *inicial contendrá el nombre de un directorio).
- Cuando no hay segundo '/', copia *camino en *inicial sin el primer '/' (entonces *inicial contendrá el nombre de un fichero).
- La función debe devolver un valor que indique si en *inicial hay un nombre de directorio o un nombre de fichero. [2]

[2] Podemos utilizar una variable llamada tipo, pasada por referencia, para guardar el tipo y así la función retorna sólo el error (en caso de que el camino esté vacío **o no empiece por '/'**). Otras posibilidades, sin esa variable, son por ejemplo, devolver 0 (si es fichero) o 1 (si es directorio), o si declaráis la función de tipo unsigned char podéis devolver 'f' o 'd'.

- Guarda en *final el resto de *camino a partir del segundo '/' **inclusive** (en caso de directorio, porque en caso de fichero no guarda nada).

Para tratamientos con cadenas de caracteres, se pueden utilizar las funciones declaradas en *string.h*: **strcpy()** copia toda una cadena, **strncpy()** copia un determinado nº de caracteres de una cadena, **strchr()** localiza la 1ª aparición de un determinado carácter y **strtok()** que trocea una cadena en tokens utilizando uno o varios delimitadores.

- Ejemplos:
  - *camino: "/dir1/dir2/fichero"
    - *inicial: "dir1"
    - *tipo: 'd'
    - *final: "/dir2/fichero"
  - *camino: "/dir/"
    - *inicial: "dir"
    - *tipo: 'd'
    - *final: "/"
  - *camino: "/fichero"
    - *inicial: "fichero"
    - *tipo: 'f'
    - *final = "" // '\0'

2) int buscar_entrada(const char *_camino_parcial_, unsigned int *_p_inodo_dir_, unsigned int *_p_inodo_, unsigned int *_p_entrada_, char _reservar_, unsigned char _permisos_);

Esta función nos buscará una determinada entrada (la parte *inicial del *camino_parcial que nos devuelva extraer_camino()) entre las entradas del inodo correspondiente a su directorio padre.
Dada una cadena de caracteres (*_camino_parcial_) y el nº de inodo del directorio padre (*_p_inodo_dir_) donde buscar la entrada en cuestión, obtiene:

- El número de inodo (*_p_inodo_) al que está asociado el nombre buscado
- El número de entrada (*_p_entrada_) dentro del inodo *p_inodo_dir que lo contiene

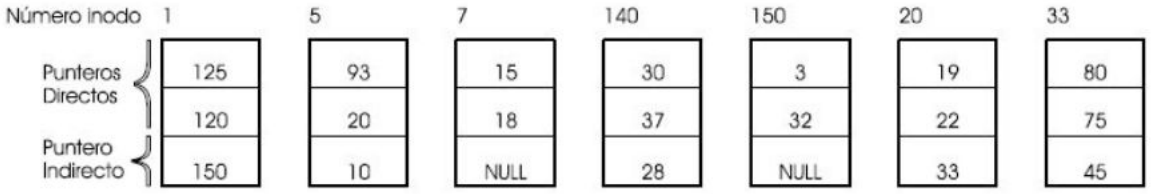
Lo más sencillo es implementar *buscar_entrada*() con **llamadas recursivas** a sí misma con la ayuda de la función *extraer_camino*().

- Ejemplo:

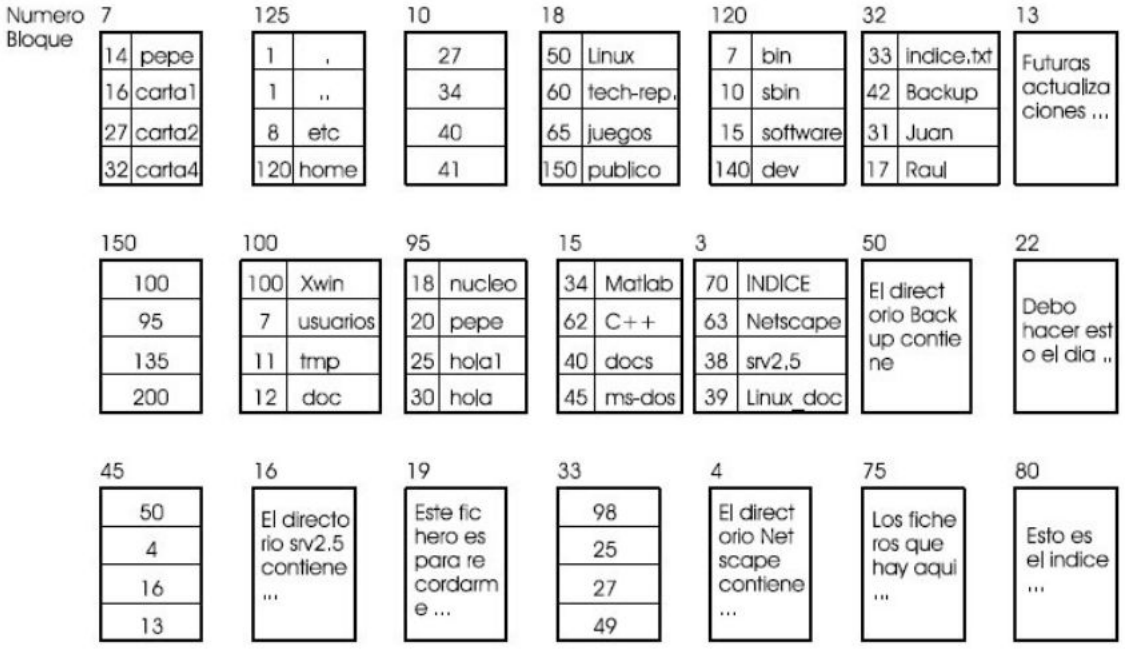
  Veamos cómo funcionaría para obtener el nº de inodo del fichero indicado por **"/usuarios/publico/indice.txt"**. En este ejemplo BLOCKSIZE=256, sizeof(struct entrada)= 64, así que tenemos 4 entradas de directorio por bloque. El struct inodo se compone, por simplicidad de 2 punteros directos y 1 indirecto. En este ejemplo, el inodo raíz está ubicado en la posición 1 del array de inodos (SB.posInodoRaiz = 1).

  En la zona de datos tenemos bloques de 3 tipos de datos:

  – Bloques de datos: unsigned char buffer [BLOCKSIZE]
  – Bloques de punteros: unsigned int punteros[BLOCKSIZE/sizeof(unsigned int)]
  – Bloques de entradas de directorio: struct entrada entradas[BLOCKSIZE/sizeof(struct entrada)]

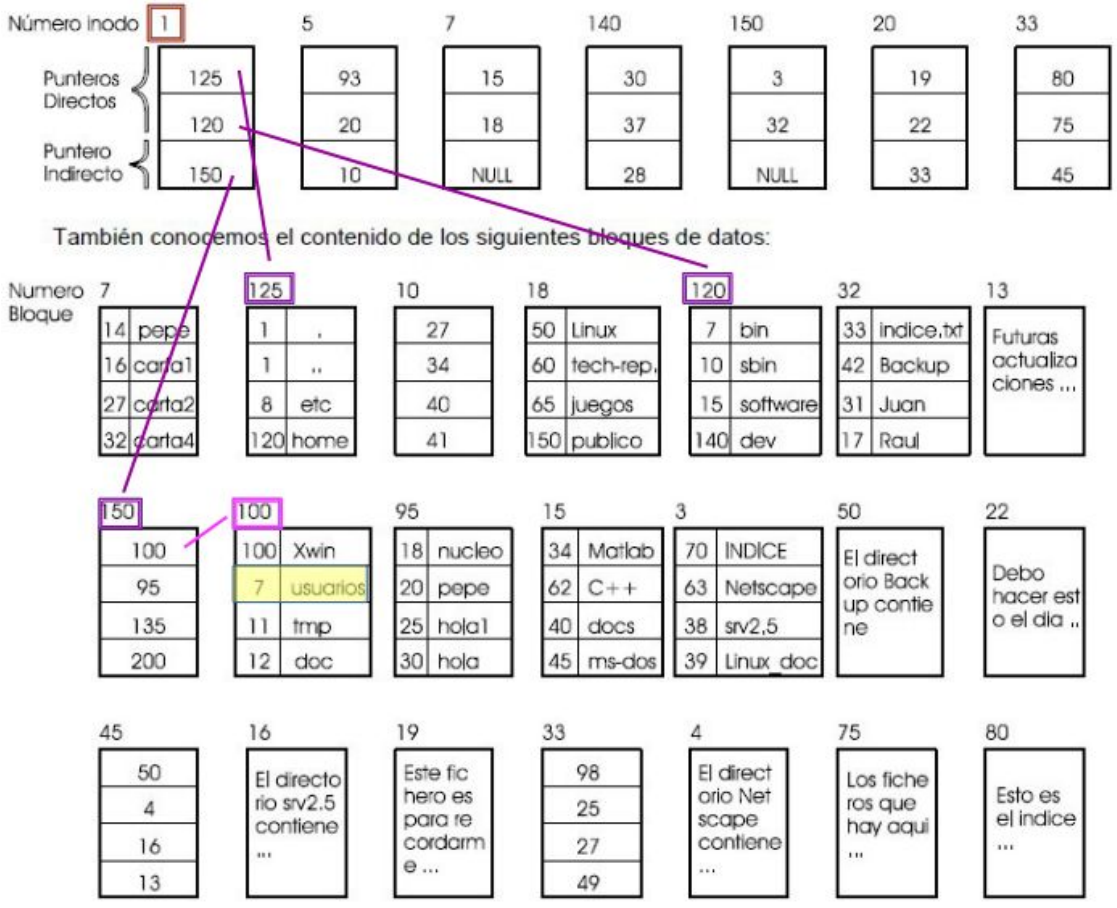


También conocemos el contenido de los siguientes bloques de datos:



  - 1ra llamada recursiva a buscar_entrada():
    - Valores de entrada::
      - *camino_parcial = "/usuarios/publico/indice.txt"

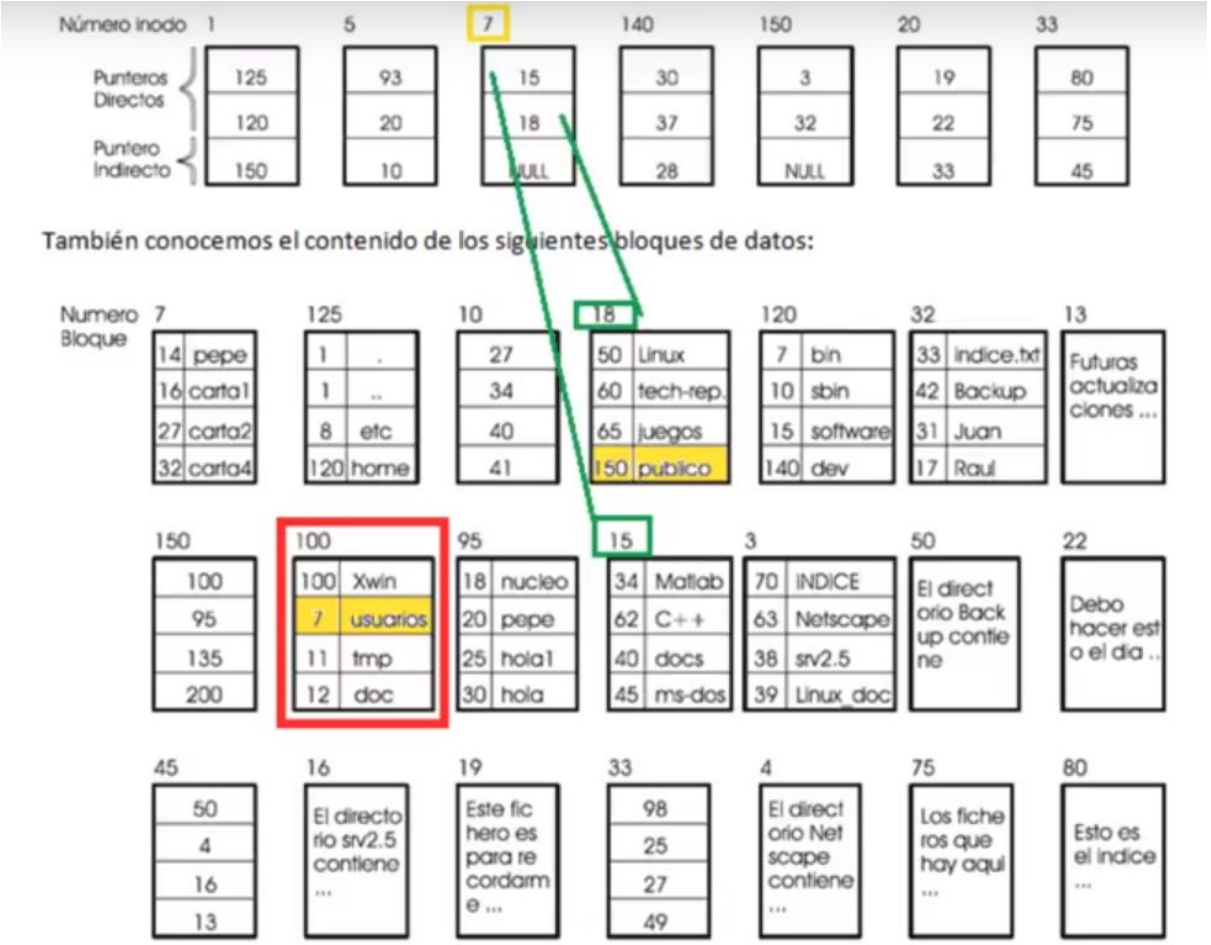
- *p_inodo_dir = 1 (número de inodo del directorio raíz "/")
- Tras llamar a extraer_camino():
  - *inicial = ”usuarios”
  - *final = “/publico/indice.txt”
- Objetivo:
  - obtener el nº de inodo (*p_inodo) asociado al nombre "usuarios" dentro del directorio "/": 7
  - y el nº de entrada correspondiente (*p_entrada)[3]: 9



- 2ª llamada recursiva a buscar_entrada():
  - Valores de entrada::
    - *camino_parcial = "/publico/indice.txt" (lo que era el valor de *final en la anterior llamada)
    - *p_inodo_dir = 7 (número de inodo de "/usuarios/", lo que era el valor de *p_inodo en la anterior llamada)
  - Tras llamar a extraer_camino():
    - *inicial = ”publico”
    - *final = “/indice.txt”
  - Objetivo:

[3] Empezamos a contar por la 0

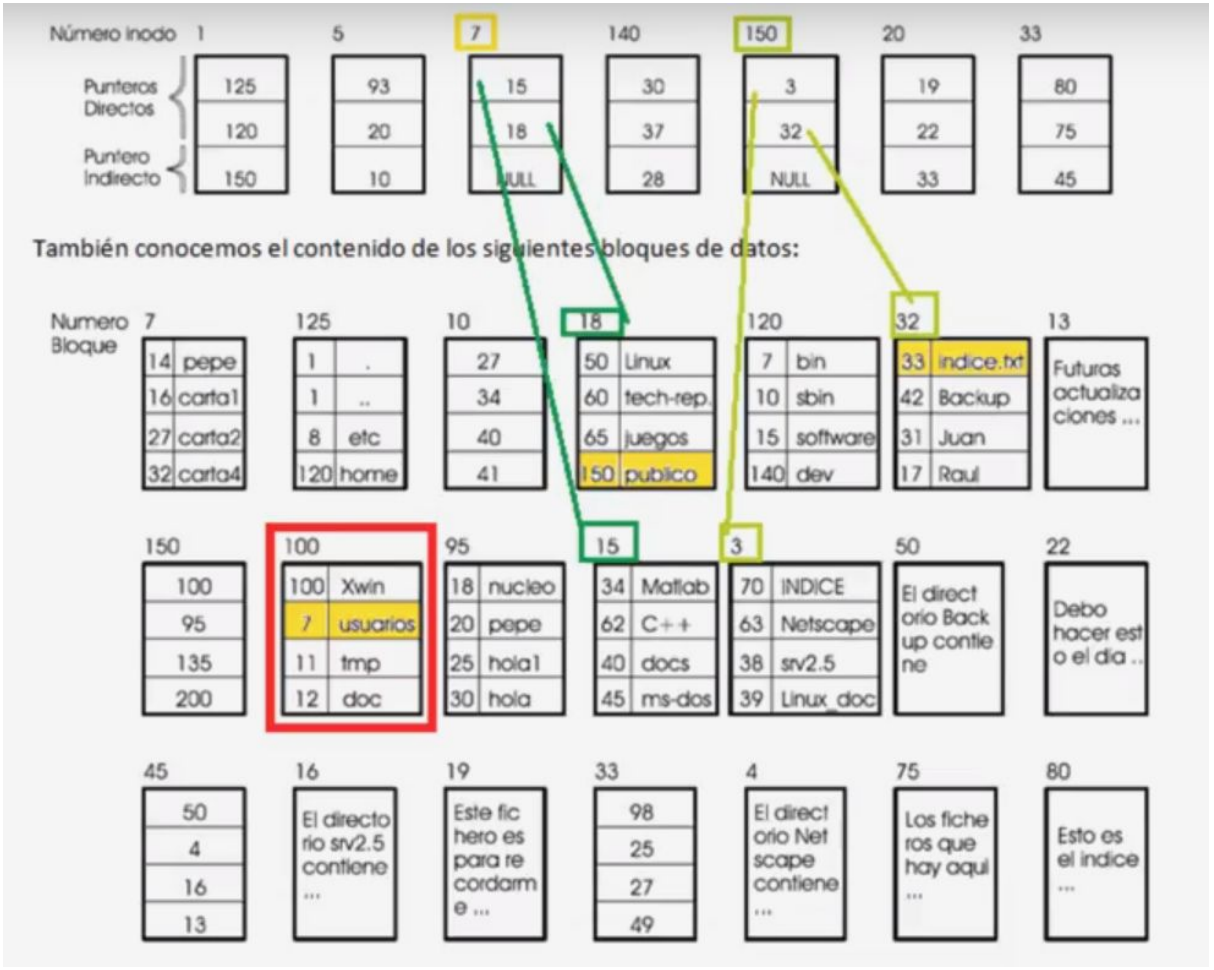
- obtener el nº de inodo (*p_inodo) asociado al nombre "publico" dentro del directorio "usuarios": 150
- y el nº de entrada correspondiente (*p_entrada)[4]: 8



- 3ª llamada recursiva a buscar_entrada():
  - Valores de entrada::
    - *camino_parcial = "/indice.txt" (lo que era el valor de *final en la anterior llamada)
    - *p_inodo_dir = 150 (número de inodo de "publico" dentro del directorio “usuarios”, lo que era el valor de *p_inodo en la anterior llamada)
  - Tras llamar a extraer_camino():
    - *inicial = ”indice.txt”
    - *final = “”
  - Objetivo:
    - obtener el nº de inodo (*p_inodo) asociado al nombre "indice.txt" dentro del directorio "publico": 33

[4] Empezamos a contar por la 0

- y el nº de entrada correspondiente (*p_entrada)[5]: 4



La misma función nos puede servir tanto para sólo **consultar** si reservar=0 (llamadas desde de *mi_unlink()*, *mi_dir()*, *mi_chmod()*, *mi_stat()*, *mi_read*() y *mi_write()* de la capa de directorios) como para consultar y **crear una entrada de directorio** si reservar=1, cuando ésta no exista (llamadas desde *mi_creat()* y *mi_link*() de la capa de directorios).

El modo sólo es relevante en caso de crear (es ignorado en caso contrario).

Se puede acceder a las entradas de directorio con llamadas a *mi_read_f*()[6]; la función *strcmp()* nos puede ser útil para saber si hemos dado con la que buscamos.

En caso de crear, hay que basarse, principalmente, en las funciones reservar_inodo() y mi_write_f().

[5] Empezamos a contar por la 0
[6] Lo adecuado sería acceder al dispositivo para cargar un bloque de entradas en un buffer sólo cuando sea necesario, es decir no volver a leer del dispositivo si aún tenemos entradas por explorar en un buffer

En caso de que *camino_parcial = "/" (directorio raíz) la función devolverá: *p_inodo= SB.posInodoRaiz; *p_entrada=0.

Hay que controlar, como mínimo, los siguientes errores:

- que la llamada a extraer_camino() dé error
- que el directorio al que apunta p_inodo_dir no tenga permisos de lectura cuando hacemos la llamada en modo lectura
- que el directorio al que apunta p_inodo_dir no tenga permisos de escritura cuando hacemos la llamada en modo escritura
- que el directorio o fichero no exista cuando hacemos la llamada en modo lectura
- que el directorio o fichero ya exista cuando hacemos la llamada en modo escritura **(nosotros consideraremos que los directorios intermedios han de existir!!!)** [7]
- que se permita crear ficheros/directorios dentro de un fichero

Todos los posibles errores de esta función se han de codificar diferenciadamente.

Ejemplo de algoritmo recursivo:

```
funcion buscar_entrada (*camino_parcial: cadena,*p_inodo_dir: ent, *p_inodo: ent, *p_entrada: ent, reservar: bool, permisos: car) devolver ent

  si (es el directorio raíz) entonces  //camino_parcial es "/"
     *p_inodo:=SB.posInodoRaiz  //nuestra raiz siempre estará asociada al inodo 0
     *p_entrada:=0
     devolver 0
  fsi

  entrada: struct entrada
  inicial[sizeof(entrada.nombre)]: car
  final[sizeof(strlen(camino_parcial))]: car
  extraer_camino (camino_parcial, inicial, final, &tipo)

  si error al extraer camino entonces devolver ERROR_EXTRAER_CAMINO  fsi

  //buscamos la entrada cuyo nombre se encuentra en inicial
  leer_inodo( *p_inodo_dir, &inodo_dir)
```

[7] El comando de Linux **mkdir** tiene una opción bastante útil que permite crear un árbol de directorios completo que no existe, mediante la opción *-p*:
$ mkdir -p /home/ejercicios/prueba/uno/dos/tres

```
si inodo_dir no tiene permisos de lectura entonces
   devolver ERROR_PERMISO_LECTURA
fsi

//el buffer de lectura puede ser un struct tipo entrada
//o bien un array de las entradas que caben en un bloque, para optimizar la lectura en RAM
calcular cant_entradas_inodo  //cantidad de entradas que contiene el inodo
num_entrada_inodo := 0  //nº de entrada inicial
si cant_entradas_inodo > 0 entonces
   leer entrada //previamente inicializar el buffer de lectura con 0s
   mientras ((num_entrada_inodo < cant_entradas_inodo) y (inicial ≠ entrada.nombre)) hacer
      num_entrada_inodo++
      leer siguiente entrada [8]  //previamente inicializar el buffer de lectura con 0s
   fmientras
fsi
si (num_entrada_inodo = cant_entradas_inodo) y (inicial ≠ entrada.nombre) entonces
//la entrada no existe
   seleccionar(reservar)
      caso 0:  //modo consulta. Como no existe retornamos error
         devolver ERROR_NO_EXISTE_ENTRADA_CONSULTA
      caso 1:  //modo escritura.
          //Creamos la entrada en el directorio referenciado por *p_inodo_dir
         //si es fichero no permitir escritura
         si inodo_dir.tipo = 'f' entonces
            devolver ERROR_NO_SE_PUEDE_CREAR_ENTRADA_EN_UN_FICHERO
         fsi
        //si es directorio comprobar que tiene permiso de escritura
         si inodo_dir no tiene permisos de escritura entonces
            devolver ERROR_PERMISO_ESCRITURA
         si_no
            copiar inicial en el nombre de la entrada
            si tipo = 'd' entonces
               si final es igual a "/" entonces [9]
                  reservar un inodo como directorio y asignarlo a la entrada
               si_no //cuelgan más diretorios o ficheros
                   devolver ERROR_NO_EXISTE_DIRECTORIO_INTERMEDIO
               fsi
            si_no //es un fichero
               reservar un inodo como fichero y asignarlo a la entrada
            fsi
            escribir la entrada
```

[8] No habría que acceder al disco si la siguiente entrada está en el mismo bloque que la anterior y por tanto ya a tenemos en un buffer

[9] Recordad que las comparaciones de cadenas se hacen con la función strcmp() y las asignaciones con strcpy()

```
            si error de escritura entonces
               si se había reservado un inodo para la entrada entonces  //entrada.inodo != -1
                  liberar el inodo
               fsi
               devolver EXIT_FAILURE
            fsi
         fsi
      fseleccionar
   fsi
   si hemos llegado al final del camino  entonces
      si (num_entrada_inodo < cant_entradas_inodo) && (reservar=1) entonces
      //modo escritura y la entrada ya existe
         devolver ERROR_ENTRADA_YA_EXISTENTE
      fsi
      // cortamos la recursividad
      asignar a *p_inodo el numero de inodo del directorio/fichero creado/leido
      asignar a *p_entrada el número de su entrada dentro del último directorio que lo contiene
      devolver EXIT_SUCCESS
   si_no
      asignamos a *p_inodo_dir el puntero al inodo que se indica en la entrada;
      devolver buscar_entrada (final, p_inodo_dir, p_inodo, p_entrada, reservar, permisos)
   fsi
ffuncion
```

Se recomienda utilizar #define para asociar un número negativo a cada símbolo de error y realizar una función auxiliar para imprimir los mensajes de los diferentes errores, pasándole el número correspondiente:

void mostrar_error_buscar_entrada(int error);

Ejemplo de asociación de símbolos en directorios.h:

```
#define ERROR_CAMINO_INCORRECTO -1
#define ERROR_PERMISO_LECTURA -2
#define ERROR_NO_EXISTE_ENTRADA_CONSULTA -3
#define ERROR_NO_EXISTE_DIRECTORIO_INTERMEDIO -4
#define ERROR_ENTRADA_YA_EXISTENTE -6
#define ERROR_NO_SE_PUEDE_CREAR_ENTRADA_EN_UN_FICHERO -7
#define ERROR_PERMISO_ESCRITURA -5
```

Ej de función para mostrar los errores de buscar_entrada(), en directorios.c

```
void mostrar_error_buscar_entrada(int error) {
   // fprintf(stderr, "Error: %d\n", error);
   switch (error) {
   case -1: fprintf(stderr, "Error: Camino incorrecto.\n"); break;
   case -2: fprintf(stderr, "Error: Permiso denegado de lectura.\n"); break;
   case -3: fprintf(stderr, "Error: No existe el archivo o el directorio.\n"); break;
   case -4: fprintf(stderr, "Error: No existe algún directorio intermedio.\n"); break;
   case -5: fprintf(stderr, "Error: Permiso denegado de escritura.\n"); break;
   case -6: fprintf(stderr, "Error: El archivo ya existe.\n"); break;
   case -7: fprintf(stderr, "Error: No es un directorio.\n"); break;
   }
}
```

TESTS DE PRUEBA

Podéis hacer un programa o adaptar leer_sf.c para hacer algunas pruebas de creación de ficheros y directorios con buscar_entrada() y detectar algunos errores[10]:

```
#include "directorios.h"

void mostrar_buscar_entrada(char *camino, char reservar){
 unsigned int p_inodo_dir = 0;
 unsigned int p_inodo = 0;
 unsigned int p_entrada = 0;
 int error;
 printf("\ncamino: %s, reservar: %d\n", camino, reservar);
 if ((error = buscar_entrada(camino, &p_inodo_dir, &p_inodo, &p_entrada, reservar, 6)) < 0) {
   mostrar_error_buscar_entrada(error);
 }
 printf("**********************************************************************\n");
 return;
}

int main(int argc, char **argv){
 if (argc!=2) {
```

[10] Los errores relacionados con permisos los testearemos en el siguiente nivel

```
   fprintf(stderr, "Sintaxis: pruebas_buscar_entrada <nombre_dispositivo>\n");
   exit(-1);
  }
  //montamos el dispositivo
  if(bmount(argv[1])<0) return -1;

//Mostrar creación directorios y errores
 mostrar_buscar_entrada("pruebas/", 1); //ERROR_CAMINO_INCORRECTO
 mostrar_buscar_entrada("/pruebas/", 0); //ERROR_NO_EXISTE_ENTRADA_CONSULTA
 mostrar_buscar_entrada("/pruebas/docs/", 1); //ERROR_NO_EXISTE_DIRECTORIO_INTERMEDIO
 mostrar_buscar_entrada("/pruebas/", 1); // creamos /pruebas/
 mostrar_buscar_entrada("/pruebas/docs/", 1); //creamos /pruebas/docs/
 mostrar_buscar_entrada("/pruebas/docs/doc1", 1); //creamos /pruebas/docs/doc1
 mostrar_buscar_entrada("/pruebas/docs/doc1/doc11", 1);
//ERROR_NO_SE_PUEDE_CREAR_ENTRADA_EN_UN_FICHERO
 mostrar_buscar_entrada("/pruebas/", 1); //ERROR_ENTRADA_YA_EXISTENTE
 mostrar_buscar_entrada("/pruebas/docs/doc1", 0); //consultamos /pruebas/docs/doc1
 mostrar_buscar_entrada("/pruebas/docs/doc1", 1); //creamos /pruebas/docs/doc1
 mostrar_buscar_entrada("/pruebas/casos/", 1); //creamos /pruebas/casos/
 mostrar_buscar_entrada("/pruebas/docs/doc2", 1); //creamos /pruebas/docs/doc2

 bumount(argv[1]);
}
```

Resultado de la ejecución con el sistema de ficheros recién inicializado [11]:

```
camino: pruebas/, reservar: 1
Error: Camino incorrecto.
********************************************************************

camino: /pruebas/, reservar: 0
[buscar_entrada()→ inicial: pruebas, final: /, reservar: 0]
Error: No existe el archivo o el directorio.
********************************************************************

camino: /pruebas/docs/, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /docs/, reservar: 1]
Error: No existe algún directorio intermedio.
```

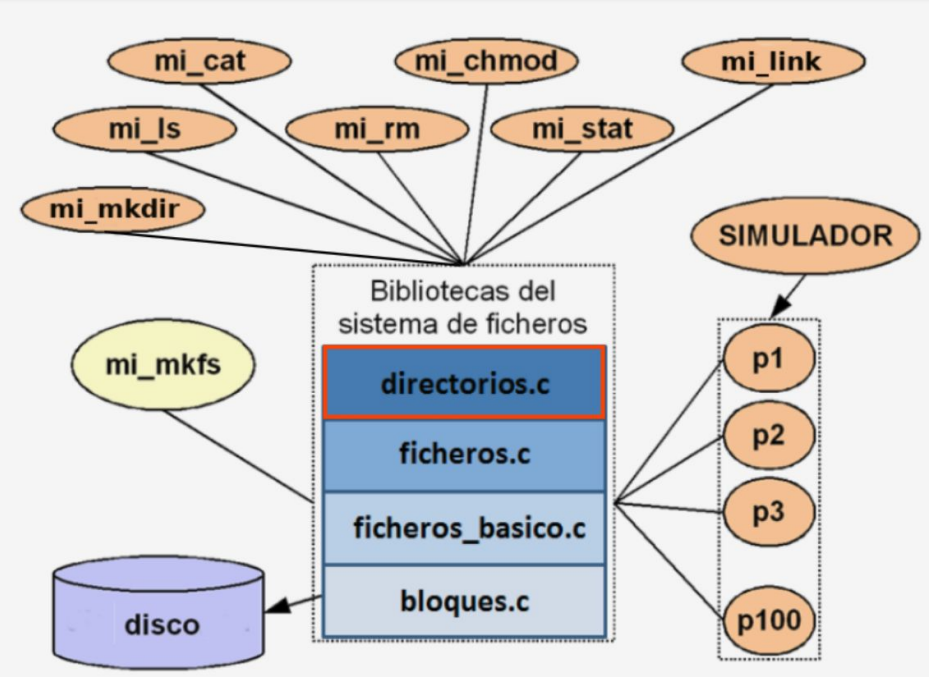
[11] Aquí ya se pueden omitir los fprintf() de traducir_bloque_inodo()

```
********************************************************************

camino: /pruebas/, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /, reservar: 1]
[buscar_entrada()→ reservado inodo 1 tipo d con permisos 6 para pruebas]
[buscar_entrada()→ creada entrada: pruebas, 1]
********************************************************************

camino: /pruebas/docs/, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /docs/, reservar: 1]
[buscar_entrada()→ inicial: docs, final: /, reservar: 1]
[buscar_entrada()→ reservado inodo 2 tipo d con permisos 6 para docs]
[buscar_entrada()→ creada entrada: docs, 2]
********************************************************************

camino: /pruebas/docs/doc1, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /docs/doc1, reservar: 1]
[buscar_entrada()→ inicial: docs, final: /doc1, reservar: 1]
[buscar_entrada()→ inicial: doc1, final: , reservar: 1]
[buscar_entrada()→ reservado inodo 3 tipo f con permisos 6 para doc1]
[buscar_entrada()→ creada entrada: doc1, 3]
********************************************************************

camino: /pruebas/docs/doc1/doc11, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /docs/doc1/doc11, reservar: 1]
[buscar_entrada()→ inicial: docs, final: /doc1/doc11, reservar: 1]
[buscar_entrada()→ inicial: doc1, final: /doc11, reservar: 1]
[buscar_entrada()→ inicial: doc11, final: , reservar: 1]
Error: No es un directorio.
********************************************************************

camino: /pruebas/, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /, reservar: 1]
Error: El archivo ya existe.
********************************************************************

camino: /pruebas/docs/doc1, reservar: 0
[buscar_entrada()→ inicial: pruebas, final: /docs/doc1, reservar: 0]
[buscar_entrada()→ inicial: docs, final: /doc1, reservar: 0]
[buscar_entrada()→ inicial: doc1, final: , reservar: 0]
********************************************************************

camino: /pruebas/docs/doc1, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /docs/doc1, reservar: 1]
[buscar_entrada()→ inicial: docs, final: /doc1, reservar: 1]
[buscar_entrada()→ inicial: doc1, final: , reservar: 1]
Error: El archivo ya existe.
```

```
**********************************************************************

camino: /pruebas/casos/, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /casos/, reservar: 1]
[buscar_entrada()→ inicial: casos, final: /, reservar: 1]
[buscar_entrada()→ reservado inodo 4 tipo d con permisos 6 para casos]
[buscar_entrada()→ creada entrada: casos, 4]
**********************************************************************

camino: /pruebas/docs/doc2, reservar: 1
[buscar_entrada()→ inicial: pruebas, final: /docs/doc2, reservar: 1]
[buscar_entrada()→ inicial: docs, final: /doc2, reservar: 1]
[buscar_entrada()→ inicial: doc2, final: , reservar: 1]
[buscar_entrada()→ reservado inodo 5 tipo f con permisos 6 para doc2]
[buscar_entrada()→ creada entrada: doc2, 5]
**********************************************************************
```

Nivel 8: directorios.c {mi_creat(), mi_dir(), mi_chmod(), mi_stat()}, y mi_mkdir.c, [mi_touch.c], mi_ls.c, mi_chmod.c, mi_stat.c

En este nivel crearemos algunas funciones de la capa de directorios y los programas correspondientes (comandos) que permiten la ejecución de tales funcionalidades desde la consola. Todas las funciones llamarán a buscar_entrada() para obtener el nº de inodo. Varias de ellas van ligadas a la función correspondiente de la capa de ficheros, a la cual le pasan ese nº de inodo.



directorios.c:	ficheros.c:
extraer_camino() buscar_entrada() _____ _	
mi_creat() **mi_dir()**	
mi_chmod() mi_stat()	mi_chmod_f() mi_stat_f()
mi_write() mi_read()	mi_write_f() mi_read_f()
mi_link() mi_unlink()	mi_truncar_f()

Tabla con la correlación de funciones de la capa de directorios con las de la capa de ficheros

A continuación vamos a ir viendo (en este nivel y en los 2 siguientes) cada una de las funciones restantes de la capa de directorios junto con cada uno de los programas externos (comandos) que las llamarán.
Todos los programas externos tienen que incluir el fichero **directorios.h**, y montar y desmontar el dispositivo virtual.

1) Creación de ficheros y directorios

1a) mi_mkdir.c

Sintaxis: `./mi_mkdir <disco> <permisos> </ruta>`

Programa (comando) que crea un fichero o directorio, llamando a la función mi_creat(). Dependiendo de si la ruta acaba en / o no se estará indicando si hay que crear un directorio o un fichero.

Hay que comprobar que permisos sea un nº válido (0-7).

Vuestro sistema de ficheros no ha de permitir crear un directorio o fichero dentro de un fichero!!! (eso lo ha de controlar buscar_entrada()).

Opcionalmente se puede crear el comando **mi_touch**, (programa adicional mi_touch.c) para crear un fichero, separando la funcionalidad de mi_mkdir que se limitaría a crear directorios, comprobando la sintaxis de camino acabado en '/'.

1b) int mi_creat(const char *camino, unsigned char permisos);

Función de la capa de directorios que crea un fichero/directorio y su entrada de directorio.
Se basa, principalmente, en la función buscar_entrada() con reservar=1.
(Realmente habría que leer el superbloque para pasarle la posición del inodo del directorio raíz, aunque por simplicidad podemos suponer directamente que p_inodo_dir es 0).
Los directorios intermedios han de existir!!! (eso lo controla buscar_entrada())
Otros posibles errores que devolvería al usuario a través de buscar_entrada():

- Que algún directorio no tenga permiso de lectura
- Que el directorio padre no tenga permiso de escritura
- Que la entrada ya exista

2)Listado del contenido de un directorio

2a) mi_ls.c

Sintaxis: `./mi_ls <disco> </ruta_directorio>`

Programa (comando) que lista el contenido de un directorio (**nombres** de las entradas), llamando a la función mi_dir() de la capa de directorios, que es quien construye el buffer[1] que mostrará mi_ls.c. **Indicaremos el total de entradas.**

Mejoras opcionales:

- Listar no solo los nombres sino varios datos de cada fichero o directorio (tipo, permisos, mtime, tamaño) tabulados, en tal caso hay que **imprimir las cabeceras**. En mi_dir() se ampliaría el contenido del buffer con esos datos.

  Ejemplo:

```
$./mi_ls disco  /di1/dir11/
Total: 2
Tipo    Permisos        mTime                   Tamaño          Nombre
---------------------------------------------------------------------------------------
--------
f       rw-             2018-04-18 13:59:45     0               fic111
f       rw-             2018-04-18 14:01:09     0               fic112
```

- Utilizar colores para diferenciar el listado de un fichero del de un directorio (la información del color la aportaría mi_dir() al buffer).
- Admitir el comando **mi_ls** también para ficheros[2] y en tal caso mostrar los datos del mismo. No habrá total de entradas. Habría que modificar mi_dir() para que, en vez de ir leyendo el inodo asociado a cada entrada, incorpore al buffer los datos del inodo correspondiente al fichero.

```
$./mi_ls disco  /di1/dir11/fic111
Tipo    Permisos        mTime                   Tamaño          Nombre
---------------------------------------------------------------------------------------
--------
f       rw-             2018-04-18 14:01:10     0               fic111
```

[1] Si vamos a listar los datos de cada entrada en una linea, podemos suponer que cada línea tiene 100 caracteres y prever un máximo de 1000 lineas:

```
#define TAMFILA 100
#define TAMBUFFER (TAMFILA*1000)
//suponemos un máx de 1000 entradas, aunque debería ser SB.totInodos
```

[2] En tal caso, en la llamada a mi_dir() desde mi_ls.c, podéis utilizar un parámetro adicional de tipo, basado en la sintaxis del camino (acabado en '/' para directorios). Para obtener el valor de ese parámetro basta examinar la sintanxis del último carácter del camino:

```
 if(camino[strlen(camino)-1]=='/') //es un directorio
```

Cuando dentro de mi_dir() se lee el inodo se tendría que comparar ese parámetro con el tipo guardado en el inodo, y si no coincide indicar "Error: la sintaxis no concuerda con el tipo" y salir.

- Utilizar una opción (parámetro) para distinguir entre mostrar formato simple y expandido de mi_ls

2b) `int mi_dir(const char *camino, char *buffer);` o
`int mi_dir(const char *camino, char *buffer, char tipo);`

Función de la capa de directorios que pone el contenido del directorio en un *buffer* de memoria (el nombre de cada entrada puede venir separado por '|' o por un tabulador) y devuelve el número de entradas. Implica leer de forma secuencial el contenido de un inodo de tipo directorio, con mi_read_f() leyendo sus entradas.[3]

Buscamos la entrada correspondiente a **camino* para **comprobar que existe** y leemos su inodo, comprobando que se trata de un **directorio** y que tiene **permisos de lectura**.
Para cada entrada concatenamos (mediante la función **strcat()**) su nombre al buffer con un separador.

Opcionalmente podemos leer también el inodo asociado a cada entrada e incorporar al buffer la información acerca de su tipo, permisos, tamaño y mtime.

Para incorporar la información acerca de los permisos:

```
si (inodo.permisos & 4) entonces strcat(buffer,"r") si_no strcat(buffer,"-") fsi
si (inodo.permisos & 2) entonces strcat(buffer,"w") si_no strcat(buffer,"-") fsi
si (inodo.permisos & 1) entonces strcat(buffer,"x") si_no strcat(buffer,"-") fsi
```

Para incorporar la información acerca del tiempo:

```
struct tm *tm; //ver info: struct tm
char tmp[100];
tm = localtime(&inodo.mtime);
sprintf(tmp,"%d-%02d-%02d %02d:%02d:%02d\t",tm->tm_year+1900,
        tm->tm_mon+1,tm->tm_mday,tm->tm_hour,tm->tm_min,tm->tm_sec);
strcat(buffer,tmp);
```

Si queremos ampliar la utilidad de mi_dir() para aplicarla también a ficheros, podemos añadir un parámetro que indique el tipo[4] y que nos lo pasará mi_ls.c, para luego poder comparar la sintaxis con el tipo real del inodo que obtendremos al leer el inodo.

[3] Aquí también se podría utilizar un buffer de n entradas, siendo n=BLOCKSIZE/sizeof(struct entrada), para no acceder al dispositivo cada vez que hay que leer una entrada
[4] **`int mi_dir(const char *camino, char *buffer, char tipo);`**

3)Cambio de permisos de un fichero o directorio

3a) `mi_chmod.c`

Sintaxis: `./mi_chmod <disco> <permisos> </ruta>`

Cambia los permisos de un fichero o directorio, llamando a la función mi_chmod().
Los permisos se indican en octal, será 4 para sólo lectura (r--), 2 para sólo escritura (-w-), 1 para sólo ejecución (--x)...
Hay que comprobar que permisos sea un nº válido (0-7).

`3b)int mi_chmod(const char *camino, unsigned char permisos);`

Buscar la entrada camino con buscar_entrada() para obtener el p_inodo. Si la entrada existe llamamos a la función correspondiente de ficheros.c pasándole el p_inodo: mi_chmod_f(p_inodo, permisos)

4)Visualización metadatos del inodo

4a) mi_stat.c

Sintaxis: `./mi_stat <disco> </ruta>`

Programa (comando) que muestra la información acerca del inodo de un fichero o directorio, llamando a la función mi_stat() de la capa de directorios.

Ejemplo de ejecución del comando stat del bash en Ubuntu:

```
$ stat bloques.c
  Fichero: bloques.c
  Tamaño: 1729       Bloques: 8          Bloque E/S: 4096   fichero regular
Dispositivo: 805h/2053d    Nodo-i: 792365      Enlaces: 1
Acceso: (0644/-rw-r--r--)  Uid: ( 1000/     uib)   Gid: ( 1000/     uib)
Acceso: 2019-03-21 18:52:41.470020991 +0100
Modificación: 2019-02-28 10:56:27.321782081 +0100
      Cambio: 2019-02-28 10:56:27.321782081 +0100
    Creación: -
```

`4b) int mi_stat(const char *camino, struct STAT *p_stat);`

Buscar la entrada camino con buscar_entrada() para obtener el p_inodo. Si la entrada existe llamamos a la función correspondiente de ficheros.c pasándole el p_inodo: mi_stat_f(p_inodo, p_stat).

Mostrar el nº de inodo.

TESTS DE PRUEBA [5]

Podéis ejecutar linea a linea o usar los scripts de test8.sh o test8touch.sh[6]

```
$ ./mi_mkfs disco 100000
$ ./mi_mkdir
Sintaxis: mi_mkdir <nombre_dispositivo> <permisos> </ruta>
$ ./mi_mkdir disco 7 / #no ha de dejar crear la raíz al usuario
$ ./mi_mkdir disco 6 dir1/
Error: Camino incorrecto.
$ ./mi_mkdir disco 6 /dir1/
[buscar_entrada()→ inicial: dir1, final: /, reservar: 1]
[buscar_entrada()→ reservado inodo 1 tipo d con permisos 6 para dir1]
[buscar_entrada()→ creada entrada: dir1, 1]
$ ./mi_mkdir disco 6 /dir1/dir11/
[buscar_entrada()→ inicial: dir1, final: /dir11/, reservar: 1]
[buscar_entrada()→ inicial: dir11, final: /, reservar: 1]
[buscar_entrada()→ reservado inodo 2 tipo d con permisos 6 para dir11]
[buscar_entrada()→ creada entrada: dir11, 2]
$ ./mi_chmod
Sintaxis: ./mi_chmod <nombre_dispositivo> <permisos> </ruta>
$ ./mi_chmod disco 1 /dir1/dir11/   #permiso sólo ejecución
[buscar_entrada()→ inicial: dir1, final: /dir11/, reservar: 0]
[buscar_entrada()→ inicial: dir11, final: /, reservar: 0]
$ ./mi_mkdir disco 6 /dir1/dir11/fic111   #o mi_touch
[buscar_entrada()→ inicial: dir1, final: /dir11/fic111, reservar: 1]
[buscar_entrada()→ inicial: dir11, final: /fic111, reservar: 1]
[buscar_entrada()→ inicial: fic111, final: , reservar: 1]
[buscar_entrada()→ El inodo 2 no tiene permisos de lectura]
Error: Permiso denegado de lectura.
$ ./mi_chmod disco 2 /dir1/dir11/  #permiso escritura
[buscar_entrada()→ inicial: dir1, final: /dir11/, reservar: 0]
[buscar_entrada()→ inicial: dir11, final: /, reservar: 0]
$ ./mi_mkdir disco 6 /dir1/dir11/fic111   #o mi_touch
[buscar_entrada()→ inicial: dir1, final: /dir11/fic111, reservar: 1]
[buscar_entrada()→ inicial: dir11, final: /fic111, reservar: 1]
[buscar_entrada()→ inicial: fic111, final: , reservar: 1]
[buscar_entrada()→ El inodo 2 no tiene permisos de lectura]
Error: Permiso denegado de lectura.
$ ./mi_chmod disco 6 /dir1/dir11/
[buscar_entrada()→ inicial: dir1, final: /dir11/, reservar: 0]
[buscar_entrada()→ inicial: dir11, final: /, reservar: 0]
$ ./mi_mkdir disco 6 /dir1/dir11/fic111   #o mi_touch
[buscar_entrada()→ inicial: dir1, final: /dir11/fic111, reservar: 1]
```

[5] Aquí ya se puede omitir la visualización de fprintf() de traducir_bloque_inodo()
[6] Ejecutado línea a línea manualmente observaréis que varían los sellos de tiempo entre sí

```
[buscar_entrada()→ inicial: dir11, final: /fic111, reservar: 1]
[buscar_entrada()→ inicial: fic111, final: , reservar: 1]
[buscar_entrada()→ reservado inodo 3 tipo f con permisos 6 para fic111]
[buscar_entrada()→ creada entrada: fic111, 3]
$ ./mi_mkdir disco 6 /dir1/dir11/fic112   #o mi_touch
[buscar_entrada()→ inicial: dir1, final: /dir11/fic112, reservar: 1]
[buscar_entrada()→ inicial: dir11, final: /fic112, reservar: 1]
[buscar_entrada()→ inicial: fic112, final: , reservar: 1]
[buscar_entrada()→ reservado inodo 4 tipo f con permisos 6 para fic112]
[buscar_entrada()→ creada entrada: fic112, 4]
$ ./mi_ls disco /
Total: 1
Tipo    Permisos      mTime                       Tamaño       Nombre
---------------------------------------------------------------------------------------
d       rw-           2020-05-13 12:28:33          64              dir1
$ ./mi_stat disco /dir1/
[buscar_entrada()→ inicial: dir1, final: /, reservar: 0]
Nª de inodo: 1
tipo: d
permisos: 6
atime: Wed 2020-05-13 12:33:00
ctime: Wed 2020-05-13 12:28:33
mtime: Wed 2020-05-13 12:28:33
nlinks: 1
tamEnBytesLog: 64
numBloquesOcupados: 1
$ ./mi_ls disco /dir1/
[buscar_entrada()→ inicial: dir1, final: /, reservar: 0]
Total: 1
Tipo    Permisos      mTime                       Tamaño       Nombre
---------------------------------------------------------------------------------------
d       rw-           2020-05-13 12:33:00          128             dir11
$ ./mi_stat disco /dir1/dir11/
[buscar_entrada()→ inicial: dir1, final: /dir11/, reservar: 0]
[buscar_entrada()→ inicial: dir11, final: /, reservar: 0]
Nª de inodo: 2
tipo: d
permisos: 6
atime: Wed 2020-05-13 12:33:00
ctime: Wed 2020-05-13 12:33:00
mtime: Wed 2020-05-13 12:33:00
nlinks: 1
tamEnBytesLog: 128
numBloquesOcupados: 1
$ ./mi_ls disco /dir1/dir11/
[buscar_entrada()→ inicial: dir1, final: /dir11/, reservar: 0]
[buscar_entrada()→ inicial: dir11, final: /, reservar: 0]
```

```
Total: 2
Tipo    Permisos      mTime                       Tamaño       Nombre
---------------------------------------------------------------------------------
f       rw-           2020-05-13 12:31:42         0            fic111
f       rw-           2020-05-13 12:33:00         0            fic112
$ ./mi_ls disco /dir1/dir12/ #Error: No existe el archivo o el directorio.
[buscar_entrada()→ inicial: dir1, final: /dir12/, reservar: 0]
[buscar_entrada()→ inicial: dir12, final: /, reservar: 0]
Error: No existe el archivo o el directorio.
$ ./mi_mkdir disco 6 /dir1/dir11/fic111  #o mi_touch #Error: El archivo ya existe
[buscar_entrada()→ inicial: dir1, final: /dir11/fic111, reservar: 1]
[buscar_entrada()→ inicial: dir11, final: /fic111, reservar: 1]
[buscar_entrada()→ inicial: fic111, final: , reservar: 1]
Error: El archivo ya existe.
$ ./mi_mkdir disco 6 /dir1/dir11/fic111/dir12/ #Error: No es un directorio
[buscar_entrada()→ inicial: dir1, final: /dir11/fic111/dir12/, reservar: 1]
[buscar_entrada()→ inicial: dir11, final: /fic111/dir12/, reservar: 1]
[buscar_entrada()→ inicial: fic111, final: /dir12/, reservar: 1]
[buscar_entrada()→ inicial: dir12, final: , reservar: 1]
Error: No es un directorio.
$ ./mi_mkdir disco 6 /dir1/dir11/dir12/fic111 #o mi_touch #Error: No existe algún
directorio intermedio
[buscar_entrada()→ inicial: dir1, final: /dir11/dir12/fic111, reservar: 1]
[buscar_entrada()→ inicial: dir11, final: /dir12/fic111, reservar: 1]
[buscar_entrada()→ inicial: dir12, final: /fic111, reservar: 1]
Error: No existe algún directorio intermedio.
$ ./mi_mkdir disco 9 /dir2/ #Error: modo inválido: <<9>>
Error:  modo inválido: <<9>>
```

Nivel 9: directorios.c {mi_read(), mi_write()}, mi_cat.c, mi_escribir.c

5)Escritura en un offset de un fichero

5a) mi_escribir.c

Sintaxis: `./mi_escribir <disco> </ruta_fichero> <texto> <offset>`

Permite escribir texto en una posición de un fichero (offset). Podéis adaptar escribir.c de la entrega parcial (incluyendo ahora directorios.h en vez de ficheros.h):

- Ha de recibir como parámetro la ruta del fichero en vez del ninodo (y ya no hay que llamar a reservar_inodo())
- Eliminar el argumento de diferentes_inodos y añadir el de offset para poder indicar desde consola dónde vamos a escribir

Hay que comprobar que se trate de un fichero, y que tenga permisos de escritura (de eso se encargará mi_write_f() de la capa de ficheros, que será llamada por mi_write() de la capa de directorios).

Mostrar la cantidad de bytes escritos.

5b)`int mi_write(const char *camino, const void *buf, unsigned int offset, unsigned int nbytes);`

Función de directorios.c para escribir contenido en un fichero. Buscaremos la entrada camino con buscar_entrada() para obtener el p_inodo. Si la entrada existe llamamos a la función correspondiente de ficheros.c pasándole el p_inodo: mi_write_f(p_inodo, buf, offset, nbytes);

Ha de devolver los bytes escritos.

Observaciones:

- Se puede optimizar la operación de escritura utilizando una **caché de directorios**: guardando el camino de la última entrada buscada y su p_inodo correspondiente (o un array de las n últimas entradas), dado que la mayoría de las lecturas/escrituras seguidas se suelen hacer sobre el mismo inodo ("principio de proximidad"), de esta forma, garantizaríamos que solo se busque la entrada si no coincide con la/s última/s buscada/s.
  - Definiríamos, en directorios.h, un struct con el camino asociado a un ninodo

```
struct UltimaEntrada{
    char camino [512];
    int p_inodo;
};
```

- Y en directorios.c una variable global:

  struct UltimaEntrada UltimaEntradaEscritura;

- Comprobaríamos si la escritura es sobre el mismo inodo:

  strcmp (camino, UltimaEntradaEscritura.camino) == 0;

- Si lo es entonces p_inodo = UltimaEntradaEscritura.p_inodo; si no llamar a buscar_entrada() y actualizar los campos de UltimaEntradaEscritura.

También podéis utilizar un array de resgistros tipo UltimaEntrada, posición 0 para lectura, posición 1 para escritura. O incluso tener una caché de más profundidad utilizando un **array de últimas entradas** de lectura o de escritura. Para la gestión de esta tabla podéis probar diferentes estrategias: FIFO, LRU, ...

6)Lectura secuencial de todo el contenido de un fichero

6a) mi_cat.c

Sintaxis: `./mi_cat <disco> </ruta_fichero>`

Programa (comando) que muestra **TODO** el contenido de un fichero (podéis adaptar leer.c de la entrega parcial para que reciba como parámetro la ruta del fichero en vez del ninodo).

Observaciones:

- Hay que comprobar que la ruta se corresponda a un fichero ya que si es un directorio no podemos hacer un cat.
- Han de coincidir los bytes leídos con el tamaño en bytes lógico del fichero y con el tamaño físico del fichero externo al que redireccionemos la lectura), y se ha de filtrar la basura.
- Utilizar una variable por ej. tambuffer para poder cambiar el tamaño del buffer de lectura sin tener que modificar todas las sentencias involucradas

6b)int mi_read(const char *camino, void *buf, unsigned int offset, unsigned int nbytes);

Función de directorios.c para leer los nbytes del fichero indicado por camino, a partir del offset pasado por parámetro y copiarlos en un buffer. Buscaremos la entrada camino con buscar_entrada() para obtener el p_inodo. Si la entrada existe llamamos a la función correspondiente de ficheros.c pasándole el p_inodo: mi_read_f(p_inodo, buf, offset, nbytes).

Ha de devolver los bytes leidos.

También se puede optimizar utilizando una caché de directorios.

TESTS DE PRUEBA [1]

```
$ ./mi_mkfs disco 100000
$ ./mi_touch disco 6 /fichero [2] #o mi_mkdir
$ ./mi_ls disco /
Total: 1
Tipo    Permisos       mTime               Tamaño  Nombre
--------------------------------------------------------------------------------
f       rw-          2018-04-25 12:58:31    0          fichero

$ ./mi_escribir  disco /fichero “hola que tal” 5120
longitud texto: 12
Bytes escritos: 12
$ ./mi_ls disco /
Total: 1
Tipo    Permisos       mTime               Tamaño  Nombre
--------------------------------------------------------------------------------
f       rw-          2018-04-25 12:58:31    5132    fichero

$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
posPrimerBloqueMB = 1
posUltimoBloqueMB = 13
posPrimerBloqueAI = 14
posUltimoBloqueAI = 3138
posPrimerBloqueDatos = 3139
posUltimoBloqueDatos = 99999
posInodoRaiz = 0
posPrimerInodoLibre = 2
cantBloquesLibres = 96859
cantInodosLibres = 24998
totBloques = 100000
totInodos = 25000
$ ./mi_chmod disco 4 /fichero
$ ./mi_escribir  disco /fichero “estoy estupendamente” 256000
longitud texto: 20
Error: Permiso denegado de escritura
Bytes escritos: 0
$ ./mi_ls disco /fichero [3]
Tipo    Permisos       mTime               Tamaño  Nombre
--------------------------------------------------------------------------------
f       r--          2018-04-25 12:58:31    5132    fichero
```

[1] Aquí ya se pueden omitir los fprintf() de buscar_entrada() salvo los errores
[2] Si no habéis implementado mi_touch podéis hacerlo con mi_mkdir
[3] Podéis ejecutarlo así si habéis implementado mi_dir() de tal manera que cree un buffer para ficheros, si no podéis ver esa información a través del padre haciendo “./mi_ls disco /”

```
# Comprobación de sellos de tiempo
$ ./mi_mkdir disco 6 /dir1/
$ ./mi_touch disco 6 /dir1/fic1 #o mi_mkdir
$ ./mi_escribir disco /dir1/fic1 hola1  256000
longitud texto: 5
Bytes escritos: 5
$ ./mi_stat disco /dir1/fic1
Nº de inodo: 3
tipo: f
permisos: 6
atime: Wed 2018-04-25 13:44:29
ctime: Wed 2018-04-25 13:44:29
mtime: Wed 2018-04-25 13:44:29
nlinks: 1
tamEnBytesLog: 256005
numBloquesOcupados: 2
$ sleep 2 #esperamos un poco para observar los sellos de tiempo
$ ./mi_escribir disco /dir1/fic1 hola2  5120 #no cambia tamenBytesLog pero sí mtime y ctime (ocupamos 1 bloque más)
longitud texto: 5
Bytes escritos: 5
$./mi_stat disco /dir1/fic1
Nº de inodo: 3
tipo: f
permisos: 6
atime: Wed 2018-04-25 13:44:29
ctime: Wed 2018-04-25 13:44:31
mtime: Wed 2018-04-25 13:44:31
nlinks: 1
tamEnBytesLog: 256005
numBloquesOcupados: 3
$ sleep 2 #esperamos un poco para observar los sellos de tiempo
$ ./mi_escribir disco /dir1/fic1 hola3  5200 #mismo bloque que offset 5120, cambia mtime pero no ctime
longitud texto: 5
Bytes escritos: 5
$ ./mi_stat disco /dir1/fic1
Nº de inodo: 3
tipo: f
permisos: 6
atime: Wed 2018-04-25 13:44:29
ctime: Wed 2018-04-25 13:44:31
mtime: Wed 2018-04-25 13:44:33
nlinks: 1
tamEnBytesLog: 256005
numBloquesOcupados: 3
$ sleep 2 #esperamos un poco para observar los sellos de tiempo
```

$ ./mi_escribir disco /dir1/fic1 hola4 256010 #cambia tamEnBytesLog, mtime y ctime
longitud texto: 5
Bytes escritos: 5
$ ./mi_stat disco /dir1/fic1
Nº de inodo: 3
tipo: f
permisos: 6
atime: Wed 2018-04-25 13:44:29
ctime: Wed 2018-04-25 13:44:35
mtime: Wed 2018-04-25 13:44:35
nlinks: 1
tamEnBytesLog: 256015
numBloquesOcupados: 3

 # Comprobación de la caché de directorios
$ ./mi_touch disco 6 /dir1/fic2 #o mi_mkdir
$ ./mi_escribir disco /dir1/fic2 "$(cat texto2.txt)" 1000
longitud texto: 3751
[mi_write() → Actualizamos la caché de escritura]
Bytes escritos: 3751
$ ./mi_cat disco /dir1/fic2 #tambuffer=BLOCKSIZE * 4
[mi_read() → Actualizamos la caché de lectura]

¿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 unimpresor (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 unade 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
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]
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.
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]

Total_leidos 4751
$ ./mi_escribir disco /dir1/fic2 “****************************” 10000**
longitud texto: 30
[mi_write() → Actualizamos la caché de escritura]
Bytes escritos: 30
$ ./mi_ls disco /dir1/
Total: 2

Tipo	Permisos	mTime	Tamaño	Nombre
f	rw-	2018-04-25 14:47:14	256015	fic1
f	rw-	2018-04-25 14:47:14	10030	fic2

$ ./mi_cat disco /dir1/fic2
[mi_read() → Actualizamos la caché de lectura]

¿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 unimpresor (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 unade 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
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]
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.
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]
****
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]

Total_leidos 10030

$ ./mi_touch disco 6 /dir1/fic3 **#o mi_mkdir**
$ ./mi_escribir_varios [4] **disco /dir1/fic3 "--texto repetido en 10 bloques--" 0** # bucle llamando a mi_escribir 10 veces con offset desplazado 1 bloque
longitud texto: 32
[mi_write() → Actualizamos la caché de escritura]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
[mi_write() → Utilizamos la caché de escritura en vez de llamar a buscar_entrada()]
Bytes escritos: 320
$ ./mi_stat disco /dir1/fic3
tipo: f
permisos: 6
atime: Wed 2018-04-25 23:40:12
ctime: Wed 2018-04-25 23:40:12
mtime: Wed 2018-04-25 23:40:12
nlinks: 1
tamEnBytesLog: 9248
numBloquesOcupados: 10
$ ./mi_cat disco /dir1/fic3 #tambuffer=BLOCKSIZE * 4
[mi_read() → Actualizamos la caché de lectura]
--texto repetido en 10 bloques----texto repetido en 10 bloques----texto repetido en 10 bloques----texto repetido en 10 bloques--
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]
--texto repetido en 10 bloques----texto repetido en 10 bloques----texto repetido en 10 bloques----texto repetido en 10 bloques--
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]
--texto repetido en 10 bloques----texto repetido en 10 bloques--
[mi_read() → Utilizamos la caché de lectura en vez de llamar a buscar_entrada()]

[4] Ver anexo

Total_leidos 9248

ANEXO

```
// mi_escribir_varios.c para probar la caché L/E
#include "directorios.h"

int main(int argc, char **argv){

 //Comprobamos sintaxis
 if (argc!=5) {
  fprintf(stderr, "Sintaxis: mi_escribir <nombre_dispositivo> </ruta_fichero> <texto> <offset>\n");
  exit(-1);
 }

 //montamos el dispositivo
 if(bmount(argv[1])<0) return -1;
 //obtenemos el texto y su longitud
 char *buffer_texto = argv[3];
 int longitud=strlen(buffer_texto);

 //obtenemos la ruta y comprobamos que no se refiera a un directorio
 if (argv[2][strlen(argv[2])-1]=='/') {
  fprintf(stderr, "Error: la ruta se corresponde a un directorio");
  exit(-1);
 }
 char *camino = argv[2];
 //obtenemos el offset
 unsigned int offset=atoi(argv[4]);
 //escribimos el texto
 int escritos=0;
 int varios = 10;
 fprintf(stderr, "longitud texto: %d\n", longitud);
 for (int i=0; i<varios; i++) {
  // escribimos varias veces el texto desplazado 1 bloque
  escritos += mi_write(camino,buffer_texto,offset+BLOCKSIZE*i,longitud);
```

```
 }
 fprintf(stderr, "Bytes escritos: %d\n", escritos);
 bumount();
}
```

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

7) Creación de enlaces físicos[1]

7a) mi_link.c

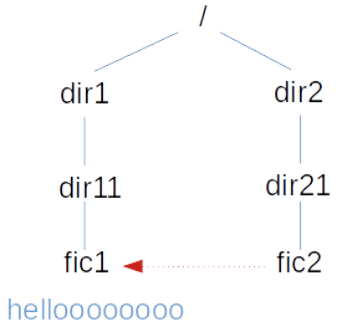
Sintaxis: ./mi_link disco /ruta_fichero_original /ruta_enlace

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 el siguiente ejemplo para crear esta estructura:




```
$ mkdir dir1
$ mkdir dir1/dir11/
$ cat > dir1/dir11/fic1 #camino1
hellooooooo [3]
$ mkdir dir2
$ mkdir dir2/dir21/
```

[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 ("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**)

[3] Pulsar Ctrl + D para acabar la edición

```
$ ln dir1/dir11/fic1 dir2/dir21/fic2  #o comando link
$ cat dir2/dir21/fic2 #ha de mostrar mismo contenido que dir1/dir11/fic1
hellooooooo
$ ls -i dir1/dir11/fic1 #la opción -i de ls nos muestra el nº de inodo
2097336 dir1/dir11/fic1
$ ls -i dir2/dir21/fic2 #comprobamos que el nº de inodo es el mismo
2097336 dir2/dir21/fic2
$ stat dir1/dir11/fic1 #comprobamos que el nº de enlaces es 2
Fichero: 'dir1/dir11/fic1'
Tamaño: 13            Bloques: 8         Bloque E/S: 4096   fichero regular
Dispositivo: 805h/2053d     Nodo-i: 2097336     Enlaces: 2
Acceso: (0664/-rw-rw-r--)  Uid: ( 1000/     uib)   Gid: ( 1000/     uib)
Acceso: 2018-05-02 08:30:52.485894470 +0200
Modificación: 2018-05-02 08:29:14.222945253 +0200
Cambio: 2018-05-02 08:30:40.085522495 +0200
Creación: -

$ 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
$ touch dir1/dir11/fic3
$ ln dir1/dir11/fic3 dir2/dir21/fic4
$ ln dir1/dir11/fic3 dir2/dir21/fic5
$ stat dir1/dir11/fic3
Fichero: 'dir1/dir11/fic3'
Tamaño: 0             Bloques: 0         Bloque E/S: 4096   fichero regular vacío
Dispositivo: 805h/2053d     Nodo-i: 2097573     Enlaces: 3
Acceso: (0664/-rw-rw-r--)  Uid: ( 1000/     uib)   Gid: ( 1000/     uib)
Acceso: 2018-05-02 08:57:42.354161048 +0200
Modificación: 2018-05-02 08:57:42.354161048 +0200
Cambio: 2018-05-02 11:11:12.616560846 +0200
Creación: -
$ 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 ninodo asociado, p_inodo1, mediante la función buscar_entrada() y comprobar que tiene **permiso de lectura**.

camino1 y camino2 han de referirse a un fichero!!! [4]

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

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

```
##############################################################################
$ ./mi_mkfs disco 100000
##############################################################################
$ ./mi_mkdir disco 6 /dir1/
$ ./mi_mkdir disco 6 /dir1/dir11/
$ ./mi_touch disco 6 /dir1/dir11/fic1 #o mi_mkdir
$ ./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: Wed 2018-05-02 11:14:35
ctime: Wed 2018-05-02 11:14:35
mtime: Wed 2018-05-02 11:14:35
nlinks: 2
tamEnBytesLog: 11
numBloquesOcupados: 1

$ ./mi_stat disco /dir2/dir21/fic2
```

```
Nº de inodo: 3
tipo: f
permisos: 6
atime: Wed 2018-05-02 11:14:35
ctime: Wed 2018-05-02 11:14:35
mtime: Wed 2018-05-02 11:14:35
nlinks: 2
tamEnBytesLog: 11
numBloquesOcupados: 1
########################################################################################
$ ./mi_link disco /dir1/dir11/fic3 /dir2/dir21/fic4  #camino1 ha de existir
Error: No existe el archivo o el directorio.
$ ./mi_touch disco 6 /dir1/dir11/fic3 #o mi_mkdir
$ ./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: Wed 2018-05-02 11:14:35
ctime: Wed 2018-05-02 11:14:35
mtime: Wed 2018-05-02 11:14:35
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** (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.

Cuando queramos borrar por ejemplo /fic1, mi_rm.c llamará a mi_unlink(), la cual llamará a:

- mi_truncar_f() para borrar la entrada fic1 del directorio raíz, a través de liberar_bloques_inodo()
- liberar_inodo(), sólo en caso de no haber enlaces, la cual llamará a liberar_bloques_inodo() para liberar tanto los bloques de datos como de punteros del fichero

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

```
$ rmdir dir2/dir21
rmdir: fallo al borrar 'dir2/dir21': El directorio no está vacío
$ rm dir2/dir21/fic2
$ stat dir1/dir11/fic1 #Hemos borrado 1 enlace
Fichero: 'dir1/dir11/fic1'
Tamaño: 13            Bloques: 8          Bloque E/S: 4096   fichero regular
Dispositivo: 805h/2053d     Nodo-i: 2097336     Enlaces: 1
Acceso: (0664/-rw-rw-r--)  Uid: ( 1000/     uib)   Gid: ( 1000/     uib)
Acceso: 2018-05-02 08:30:52.485894470 +0200
Modificación: 2018-05-02 08:29:14.222945253 +0200
Cambio: 2018-05-02 11:19:24.672635523 +0200
Creación: -
$ rm dir2/dir21/fic2
rm: no se puede borrar 'dir2/dir21/fic2': No existe el archivo o el directorio
$ rmdir dir2/dir21
rmdir: fallo al borrar 'dir2/dir21': El directorio no está vacío
$ ls -l dir2/dir21
total 0
-rw-rw-r-- 3 uib uib 0 may  2 08:57 fic4
-rw-rw-r-- 3 uib uib 0 may  2 08:57 fic5
$ rm dir2/dir21/fic4
$ rm dir2/dir21/fic5
$ rmdir dir2/dir21
$ 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.

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 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.

TESTS DE PRUEBA (continuación de los anteriores)

```
$ ./mi_rm disco /dir2/dir21/ #o mi_rmdir
Error: El directorio /dir2/dir21/ no está vacío
$ ./mi_rm disco /dir2/dir21/fic2
$ ./mi_stat disco /dir1/dir11/fic1 #Hemos borrado 1 enlace
Nº de inodo: 3
tipo: f
permisos: 6
atime: Wed 2018-05-02 12:33:40
ctime: Wed 2018-05-02 12:33:40
mtime: Wed 2018-05-02 12:33:40
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    Permisos        mTime               Tamaño  Nombre
---------------------------------------------------------------------------------------
f       rw-         2018-05-02 12:33:40     0       fic5
f       rw-         2018-05-02 12:33:40     0       fic4

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

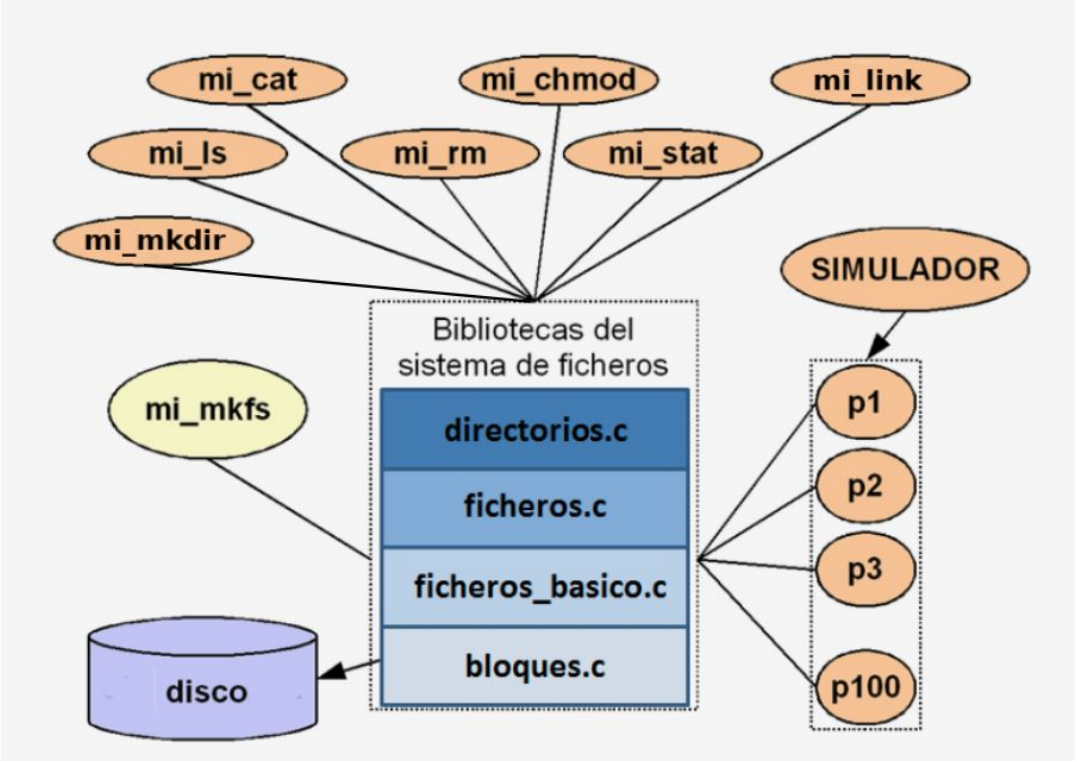
```
##############################################################################
# Comprobamos que al crear 17 subdirectorios los bloques de datos del padre son 2
# (en un bloque caben 16 entradas de directorio),
# y que al eliminar un subdirectorio el directorio padre tiene 1 bloque de datos
##############################################################################
$ ./mi_mkdir disco 6 /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: Wed 2018-05-02 12:52:45
ctime: Wed 2018-05-02 12:52:45
mtime: Wed 2018-05-02 12:52:45
nlinks: 1
tamEnBytesLog: 1088
numBloquesOcupados: 2

##############################################################################
# Listamos el directorio para ver sus subdirectorios
$ ./mi_ls disco /d1/
Total: 17
Tipo    Permisos        mTime               Tamaño  Nombre
---------------------------------------------------------------------------------------
d       rw-         2018-05-02 12:52:45     0       sd0
d       rw-         2018-05-02 12:52:45     0       sd1
d       rw-         2018-05-02 12:52:45     0       sd2
d       rw-         2018-05-02 12:52:45     0       sd3
d       rw-         2018-05-02 12:52:45     0       sd4
d       rw-         2018-05-02 12:52:45     0       sd5
d       rw-         2018-05-02 12:52:45     0       sd6
```

```
d       rw-         2018-05-02 12:52:45    0       sd7
d       rw-         2018-05-02 12:52:45    0       sd8
d       rw-         2018-05-02 12:52:45    0       sd9
d       rw-         2018-05-02 12:52:45    0       sd10
d       rw-         2018-05-02 12:52:45    0       sd11
d       rw-         2018-05-02 12:52:45    0       sd12
d       rw-         2018-05-02 12:52:45    0       sd13
d       rw-         2018-05-02 12:52:45    0       sd14
d       rw-         2018-05-02 12:52:45    0       sd15
d       rw-         2018-05-02 12:52:45    0       sd16

################################################################################
# Eliminamos el subdirectorio sd3 de d1
$ ./mi_rm disco /d1/sd3/ #o mi_rmdir
################################################################################
# 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: Wed 2018-05-02 12:52:45
ctime: Wed 2018-05-02 12:52:45
mtime: Wed 2018-05-02 12:52:45
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    Permisos        mTime               Tamaño  Nombre
--------------------------------------------------------------------------------
d       rw-         2018-05-02 12:52:45    0       sd0
d       rw-         2018-05-02 12:52:45    0       sd1
d       rw-         2018-05-02 12:52:45    0       sd2
d       rw-         2018-05-02 12:52:45    0       sd16
d       rw-         2018-05-02 12:52:45    0       sd4
d       rw-         2018-05-02 12:52:45    0       sd5
d       rw-         2018-05-02 12:52:45    0       sd6
d       rw-         2018-05-02 12:52:45    0       sd7
d       rw-         2018-05-02 12:52:45    0       sd8
d       rw-         2018-05-02 12:52:45    0       sd9
d       rw-         2018-05-02 12:52:45    0       sd10
d       rw-         2018-05-02 12:52:45    0       sd11
d       rw-         2018-05-02 12:52:45    0       sd12
d       rw-         2018-05-02 12:52:45    0       sd13
d       rw-         2018-05-02 12:52:45    0       sd14
d       rw-         2018-05-02 12:52:45    0       sd15
```

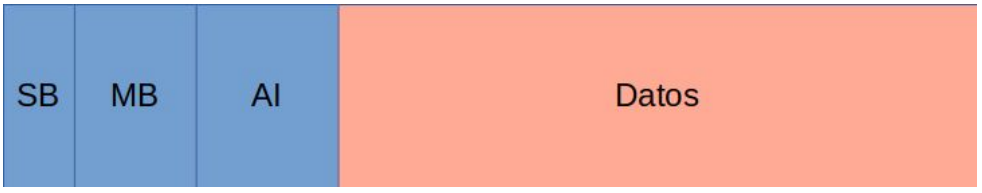
Otras funcionalidades extras

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 después de la revisió final.

Nivel 11: semáforos y exclusión mutua



Para que el sistema de ficheros pueda ser utilizado simultáneamente por más de un proceso, hay que controlar, mediante un **semáforo** (exclusión mutua es suficiente), el **acceso concurrente a los metadatos**, es decir al **superbloque**, al **mapa de bits** y al **array de inodos**.
Esta responsabilidad es del sistema de ficheros, nunca de los programas cliente.
En cambio **no es responsabilidad del sistema de ficheros controlar el acceso concurrente a los bloques de datos**, sino de aquellos programas cliente que así lo deseen.



Las **secciones críticas** que tendremos que controlar serán las porciones de código donde:

- **se reserven o liberen bloques** (afecta a los campos SB.cantBloquesLibres y al mapa de bits),
- **se reserven o liberen inodos** (afecta a los campos SB.posprimerInodoLibre, SB.cantInodosLibres)
- **y cuando se modifiquen campos de los inodos** (tipo, permisos, sellos de tiempo, nlinks, tamEnBytesLog, numBloquesOcupados, punteros)

Lo más simple es realizar un *big lock*, es decir poner **un wait al inicio de la función y un signal antes de cualquier salida de la misma**, pero hay que llevar especial cuidado con las funciones más frecuentes que son, en primer lugar el mi_read() y en segundo el mi_write() para **no serializarlas**.

Funciones de directorios.c que requieren semáforo:

- **mi_creat():** implica **reservar un inodo** para el fichero/directorio que queremos crear al llamar a buscar_entrada() y también puede implicar **reservar un bloque** al llamar a mi_write_f() para escribir la entrada de directorio, quien a su vez llama a traducir_bloque_inodo() con reservar=1.
- **mi_link():** implica **reservar un inodo** al llamar a buscar_entrada() y **liberarlo** después. También puede implicar **reservar un bloque** al llamar a mi_write_f() para escribir la entrada de directorio del enlace. Y además se modifica la información del inodo (nlinks, ctime)
- **mi_unlink():** Puede implicar **liberar un bloque** al eliminar una entrada de directorio llamando a mi_truncar_f(). Y también puede implicar **liberar un inodo** y **liberar sus bloques** si no había más enlaces. Además se modifica la información del inodo (tamEnBytesLog, nlinks, ctime)
- **mi_write():** si no se sobreescribe implica **reservar bloques** al llamar a traducir_bloque_inodo() y por tanto se modifica numBloquesOcupados y el ctime. También se modifica la información del inodo: mtime y, si el fichero crece, también tamEnBytesLog y por tanto ctime.
    - Se puede hacer más granular poniendo la sección crítica en **mi_write_f()** pero que sólo afecte a la porcion de código donde se actualiza la información del inodo (leyendo primero de nuevo el inodo y escribiéndolo justo a continuación):

        wait
        leer_inodo
        actualizar ctime, mtime (y tamEnBytesLog si ha variado el tamaño)
        escribir_inodo
        signal

      Además se ha de poner una sección critica que englobe la llamada a traducir_bloque_inodo() con reservar=1 ya que eso implica **reservar bloques**.
- **mi_read()** si no se tiene en cuenta el atime no seria necesario poner seccion crítica. **Si lo tenemos en cuenta hay que evitar serializarla:**
    - Se puede granularizar poniéndola en **mi_read_f()** pero que sólo afecte a la porcion de código donde se actualiza la información del inodo (se puede incluir la seccion crítica **al principio de la función** de tal manera que sólo se lea el inodo una vez):

        wait

leer el inodo
actualizar el atime
escribir el inodo
signal

En **mi_stat()** no hace falta semáforo.
En **mi_dir()** no hace falta semáforo si ya se utiliza en mi_read_f() cuando se actualiza atime.
En **mi_chmod()** no es necesario poner semáforo pero sí en **mi_chmod_f()** ya que se actualizan permisos y ctime**.**

Además para garantizar la consistencia del sistema de ficheros habría que llevar cuidado en que el orden de las acciones de ciertas funciones (que deberían ser atómicas) sea correcto[1], por ejemplo:

- al borrar un fichero/directorio, eliminar la entrada de directorio antes de eliminar el inodo,
- al truncar, marcar los punteros del inodo como "null" antes de liberar los bloques,
- al escribir, grabar 1º los bloques antes de modificar el tamaño del fichero

---

Utilizaremos **semáforos POSIX con nombre** (ver código semaforo_mutex_posix.h y semaforo_mutex_posix.c).

Para linkar el semáforo al programa que lo va a utilizar hay que utilizar la librería **pthread**. Ejemplo:

```
$ gcc -pthread programa.c semaforo_mutex_posix.c -o programa
```

En semaforo_mutex_posix.h se declaran las siguientes funciones:

```
sem_t *initSem();
void deleteSem();
void signalSem(sem_t *sem);
void waitSem(sem_t *sem);
```

que a su vez llaman a las funciones de <semaphore.h>: sem_open(), sem_unlink(), sem_post(), sem_wait().

El nombre de nuestro semáforo será "/mymutex" y lo inicializaremos a 1 (en semaforo_mutex.posix.h):

[1] +info: Consistencia y mantenimiento de un sistema de archivos de UNIX (págs 33 a 35)

```
#define SEM_NAME "/mymutex" // Usamos este nombre para el semáforo mutex
#define SEM_INIT_VALUE 1 // Valor inicial de los mutex */
```

En bloques.c:

- `#include "semaforo_mutex_posix.h"`.
- utilizaremos una variable global para el semáforo: `static sem_t *mutex;`
- Inicializaremos el semáforo desde bmount(): `mutex = initSem();`
- Eliminaremos el semáforo desde bumount(): `deleteSem();`
- Definiremos unas **funciones propias** para llamar a waitSem() y signalSem() (de esta manera todas las llamadas a las funciones de semaforo_mutex_posix.c estarán concentradas en bloques.c, y si cambiásemos el semáforo no habría que tocar el código del resto de programas):

```
void mi_waitSem() {
    waitSem(mutex);
}

void mi_signalSem() {
    signalSem(mutex);
}
```

En las funciones donde vayamos a definir las secciones críticas, habrá que realizar un Wait y luego incorporar un Signal **en todas las posibles salidas de la función.**

Ejemplo en mi_creat() de directorios.c:

```
int mi_creat(...) {
    mi_waitSem();
    ...
    if (error= buscar_entrada(...) <0) {
        mostrar_error_buscar_entrada(error);
        mi_signalSem();
        return -1;
    }
    mi_signalSem();
```

```
    return 0;
}
```

Esquema de las llamadas a funciones de semáforos:

semaphore.h	semaforo_mutex_posix.c	bloques.c	directorios.c/ficheros.c
sem_open() sem_unlink() sem_post() sem_wait()	initSem() deleteSem() signalSem() waitSem()	bmount() bumount() mi_signalSem() mi_waitSem()	□ □

Cómo evitar que se haga el wait dos veces (código reentrante), por ejemplo para el mi_creat() que a su vez llama a mi_read_f():

```
static unsigned int inside_sc = 0;
    ...
    void mi_waitSem(){
        if (!inside_sc) {
          waitSem(mutex);
        }
        inside_sc++;
    }
    void mi_signalSem() {
        inside_sc--;
        if (!inside_sc) {
            signalSem(mutex);
        }
    }
```

Makefile para compilar con la librería pthread:

```
CC=gcc
CFLAGS=-c -g -Wall -std=gnu99
LDFLAGS=-pthread

SOURCES=bloques.c ficheros_basico.c ficheros.c directorios.c mi_mkfs.c
leer_sf.c mi_mkdir.c mi_chmod.c mi_ls.c mi_link.c mi_escribir.c mi_cat.c mi_stat.c
mi_rm.c semaforo_mutex_posix.c #escribir.c leer.c truncar.c permitir.c
LIBRARIES=bloques.o ficheros_basico.o ficheros.o directorios.o
semaforo_mutex_posix.o
INCLUDES=bloques.h ficheros_basico.h ficheros.h directorios.h
semaforo_mutex_posix.h
PROGRAMS=mi_mkfs leer_sf mi_mkdir mi_chmod mi_ls mi_link mi_escribir
mi_cat mi_stat mi_rm #escribir.c leer.c truncar.c permitir.c

OBJS=$(SOURCES:.c=.o)

all: $(OBJS) $(PROGRAMS)

$(PROGRAMS): $(LIBRARIES) $(INCLUDES)
	$(CC) $(LDFLAGS) $(LIBRARIES) $@.o -o $@

%.o: %.c $(INCLUDES)
	$(CC) $(CFLAGS) -o $@ -c $<

.PHONY: clean
clean:
	rm -rf *.o *~ $(PROGRAMS)
```

Nivel 12: Simulación de lecturas/escrituras concurrentes

En este nivel se trata de crear un programa **simulador** (simulacion.c y su correspondiente simulacion.h), encargado de crear unos **procesos** de prueba que accedan de forma concurrente al sistema de ficheros, de modo que se pueda comprobar el correcto funcionamiento de nuestra biblioteca de funciones para más de un proceso en ejecución.

Primeramente se creará el directorio "**`simul_`*`aaaammddhhmmss`***", donde *aaaa* es el año, *mm* es el mes, *dd* es el día, *hh* es la hora, *mm* es el minuto y *ss* es el segundo de creación.
Se han de generar 100 procesos de prueba [1] cada 0,2 segundos [2]. Cada proceso creará un directorio llamado "`proceso_n`" dentro del directorio "`simul_aaaammddhhmmss`", donde n es el PID del proceso [3]. Además, dentro del directorio "`proceso_n`", cada proceso creará un fichero denominado "`prueba.dat`".
Cada 0,05 segundos [4] y un total de 50 veces, cada proceso escribirá dentro del fichero "`prueba.dat`" un registro del siguiente tipo:

---

[1] Recordad que la función fork() permite crear un nuevo proceso

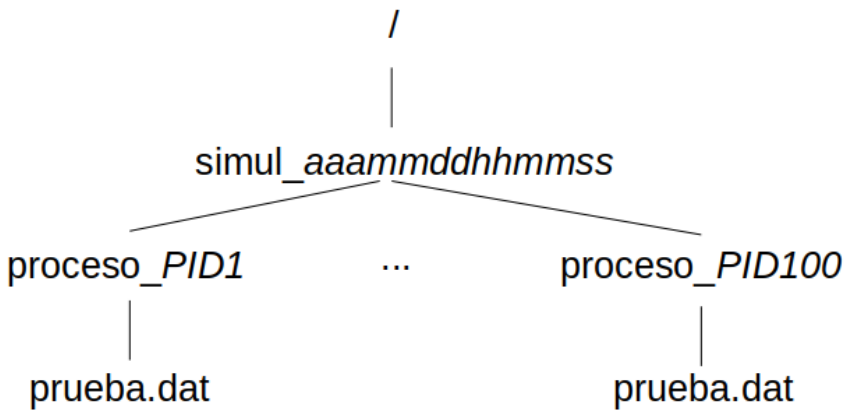
[2] Se puede provocar una espera mediante la función usleep(), especificando el tiempo en microsegundos. Hay que tener en cuenta que esta función es cancelada por la llegada de alguna señal (por ejemplo cuando un hijo acaba y se envía la señal SIGCHLD), y por tanto la ejecución de la simulación duraría menos de 100*0,2 segundos. Alternativamente se puede usar nanosleep(), especificando el tiempo en nanosegundos, y forzar la espera en caso de que llegue una señal de la siguiente manera:

```
void my_sleep(unsigned msec) { //tiempo en milisegundos
  struct timespec req, rem;
  int err;
  req.tv_sec = msec / 1000;
  req.tv_nsec = (msec % 1000) * 1000000;
  while ((req.tv_sec != 0) || (req.tv_nsec != 0)) {
    if (nanosleep(&req, &rem) == 0)
      break;
    err = errno;
    // Interrupted; continue
    if (err == EINTR) {
      req.tv_sec = rem.tv_sec;
      req.tv_nsec = rem.tv_nsec;
    }
  }
}
```

[3] Podeís concatenar cadenas de texto, mezclando variables y texto fijo, utilizando las funciones strcat() y/o sprintf().

[4] También se puede provocar esa espera mediante la función usleep() o nanosleep(),

```
//simulación.h
struct REGISTRO {
   time_t fecha; //fecha de la escritura en formato epoch
   pid_t pid; //PID del proceso que lo ha creado
   int nEscritura; //Entero con el número de escritura (de 1 a 50)
   int nRegistro; //Entero con el número del registro dentro del fichero (de 0 a REGMAX-1) [5]
};
```



El algoritmo más detallado a seguir sería el siguiente:

Comprobar sintaxis // uso: ./simulacion <disco>
Montar disco. //proceso padre
Crear el directorio de simulación en la raíz: /simul_aaaammddhhmmss/
(en caso de error desmontar el dispositivo).

Asignamos la función enterrador [6] a la señal de finalización de un hijo, SIGCHLD, la cual eliminará los hijos zombie:

```
signal(SIGCHLD, reaper);
```

Para proceso:=1 hasta proceso <=100 hacer
   Crear un proceso con fork().

[5] Esta posición se elegirá de manera aleatoria, teniendo en cuenta que el registro puede ocupar cualquier ubicación dentro del espacio de datos del fichero. De esta manera comprobaremos el correcto funcionamiento de los punteros directos e indirectos de los inodos. No importa si algún registro sobreescribe alguno de los registros anteriores.

[6] Recordemos que cuando un proceso hijo termina, el sistema guarda el PID (Identificador) y su estado (un parámetro) para dárselo a su padre. Hasta entonces el proceso finalizado entra en estado zombie. Cuando un proceso finaliza, toda la memoria y recursos asociados con dicho proceso son liberados, pero la entrada del mismo en la tabla de procesos aún existe, para cuando su padre llame a la función wait() devolverle su PID y estado.

```
Si se trata del hijo entonces //pid=0
    Montar disco. //proceso hijo [7]
    Crear el directorio del proceso, añadiendo su pid al nombre, dentro del directorio de
    simulación; (en caso de error desmontar el dispositivo).
    Crear el fichero prueba.dat dentro del directorio del proceso; (en caso de error
    desmontar el dispositivo).
    Inicializar la semilla para los números aleatorios[8]: srand(time(NULL)+getpid())
    Para cada una de las 50 operaciones de escritura i (i inicialmente=0) hacer
        Inicializar el registro:
        • registro.fecha=time(NULL);
        • registro.pid = getpid();
        • registro.nEscritura = i+1;
        • registro.nRegistro =  rand() % REGMAX; [9]
        Escribir el registro con mi_write() en registro.nRegistro * sizeof(struct registro).
        Esperar 0'05 seg para hacer la siguiente escritura.
    fpara
    Desmontar disco. //proceso hijo
    exit (0) //Necesario para que se emita la señal SIGCHLD
  fsi
  Esperar 0'2 seg para lanzar otro proceso.
fpara
//Permitir que el padre espere por todos los hijos: [10]
    while (acabados < NUMPROCESOS){
      pause()
    }
Desmontar disco. // proceso padre
exit (0) // o return 0 [11]
```

Función enterrador:

[7] El proceso principal hace el bmount() por lo que abre el fichero con el open(). Luego hace varios fork()s, y los hijos comparten los datos de los descriptores del fichero del padre. También se comparte el puntero (cambiado por el lseek()). Un proceso hace el lseek() para leer o escribir, otro hace lo mismo y lo deja cambiado, cuando el primero hace su operación de lectura o escritura accede a posiciones diferentes al lseek que hizo previamente. Hacer el bmount() nuevamente elimina ese problema de concurrencia.

[8] Para que genere números diferentes en cada ejecución para obtener los números de registro.

[9] Nuestro sistema de ficheros nos permitiría un REGMAX = (((12+256+256²+256³)-1)*BLOCKSIZE) /sizeof(struct registro) pero en la simulación lo limitaremos a 500.000 registros (valor de REGMAX)

[10] Tendremos una variable global, acabados, inicializada a 0 para llevar la cuenta del nº de procesos finalizados, y que el enterrador irá incrementado

[11] Hay discusiones a favor y en contra de la conveniencia en C de usar exit() o return para finalizar el main():
https://stackoverflow.com/questions/461449/return-statement-vs-exit-in-main
https://stackoverflow.com/questions/3463551/what-is-the-difference-between-exit-and-return/3463562

```
void reaper(){
  pid_t ended;
  signal(SIGCHLD, reaper);
  while ((ended=waitpid(-1, NULL, WNOHANG))>0) {
    acabados++;
   //Podemos testear qué procesos van acabando:
    //fprintf(stderr, "acabado: %d total acabados: %d\n", ended, acabados);
  }
}
```

Observaciones:

- Se precisa un #include <sys/wait.h> y #include <signal.h>
- Dado que en cada proceso hijo montamos y desmontamos el dispositivo virtual, tendremos que modificar las funciones bmount() y bumount() de **bloques.c**:

```
static int descriptor = 0;

int bmount(const char *camino) {
   if (descriptor > 0) {
      close(descriptor);
   }
   if ((descriptor = open(camino, O_RDWR | O_CREAT, 0666)) == -1) {
     fprintf(stderr, "Error: bloques.c → bmount() → open()\n");
   }
   if (!mutex) { //mutex == 0
   //el semáforo es único y sólo se ha de inicializar una vez en nuestro sistema (lo hace el padre)
      mutex = initSem(); //lo inicializa a 1
      if (mutex == SEM_FAILED) {
         return -1;
      }
   }
   return descriptor;
}

int bumount() {
   descriptor = close(descriptor);
   // hay que asignar el resultado de la operación a la variable ya que bmount() la utiliza
   if (descriptor == -1) {
      fprintf(stderr, "Error: bloques.c → bumount() → close(): %d: %s\n", errno, strerror(errno)");
      return -1;
   }
```

```
  deleteSem(); // borramos semaforo [12]
  return 0;
}
```

Se aconseja borrar el dispositivo virtual antes de crear otro con el mismo nombre y empezar las pruebas con 1 solo proceso y 5-10 escrituras, luego ir aumentando el nº de procesos y nº de escrituras e ir comprobando los resultados poco a poco antes de expandir la simulación a los 100 procesos y 50 escrituras.
Al final del documento tenéis un **makefile** para compilar con la librería pthread.

Ejemplo de testing:

```
*** Simulación de 3 procesos realizando cada uno 10 escrituras ***
Directorio simulación: /simul_20180516143353/
[simulación.c → Escritura 1 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 2 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 3 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 4 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 1 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 5 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 2 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 6 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 7 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 3 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 4 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 8 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 1 en /simul_20180516143353/proceso_31656/prueba.dat]
[simulación.c → Escritura 9 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 5 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 2 en /simul_20180516143353/proceso_31656/prueba.dat]
[simulación.c → Escritura 10 en /simul_20180516143353/proceso_31653/prueba.dat]
[simulación.c → Escritura 6 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 3 en /simul_20180516143353/proceso_31656/prueba.dat]
Proceso 1: Completadas 10 escrituras en /simul_20180516143353/proceso_31653/prueba.dat
[simulación.c → Escritura 7 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Acabado proceso con PID 31653, total acabados: 1
[simulación.c → acabados: 1
[simulación.c → Escritura 4 en /simul_20180516143353/proceso_31656/prueba.dat]
[simulación.c → Escritura 8 en /simul_20180516143353/proceso_31655/prueba.dat]
```

[12] La función sem_unlink(), a la que llama nuestra función deleteSem(), eliminará el semáforo nombrado por el nombre de la cadena. Si el semáforo nombrado por nombre actualmente está referenciado por otros procesos, entonces sem_unlink () no tendrá ningún efecto sobre el estado del semáforo. Si uno o más procesos tienen el semáforo abierto cuando se llama sem_unlink (), **la destrucción del semáforo se pospone hasta que todas las referencias al semáforo hayan sido destruidas por las llamadas a sem_close(), _exit() o exec**.

```
[simulación.c → Escritura 5 en /simul_20180516143353/proceso_31656/prueba.dat]
[simulación.c → Escritura 9 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 6 en /simul_20180516143353/proceso_31656/prueba.dat]
[simulación.c → Escritura 10 en /simul_20180516143353/proceso_31655/prueba.dat]
[simulación.c → Escritura 7 en /simul_20180516143353/proceso_31656/prueba.dat]
Proceso 2: Completadas 10 escrituras en /simul_20180516143353/proceso_31655/prueba.dat
[simulación.c → Acabado proceso con PID 31655, total acabados: 2
[simulación.c → acabados: 2
[simulación.c → Escritura 8 en /simul_20180516143353/proceso_31656/prueba.dat]
[simulación.c → Escritura 9 en /simul_20180516143353/proceso_31656/prueba.dat]
[simulación.c → Escritura 10 en /simul_20180516143353/proceso_31656/prueba.dat]
Proceso 3: Completadas 10 escrituras en /simul_20180516143353/proceso_31656/prueba.dat
[simulación.c → Acabado proceso con PID 31656, total acabados: 3
Total de procesos terminados: 3.
```

Una vez testeado el funcionamiento para unos pocos procesos y unas pocas escrituras podéis eliminar las impresiones entre corchetes y lanzarlo, para los 100 procesos con 50 escrituras cada una, con el comando time [13] delante para contrastar vuestro tiempo de ejecución. **Es imprescindible la impresión del nombre del directorio de simulación para la posterior verificación de las escrituras ya que lo necesitaremos como parámetro.**

```
$ ./mi_mkfs disco 100000
$ /usr/bin/time ./simulacion disco  #mostrar dir_simulacion
*** Simulación de 100 procesos realizando cada uno 50 escrituras ***
Directorio simulación: /simul_20180523133828/
Proceso 1: Completadas 50 escrituras en /simul_20180523133828/proceso_16776/prueba.dat
Proceso 2: Completadas 50 escrituras en /simul_20180523133828/proceso_16779/prueba.dat
Proceso 3: Completadas 50 escrituras en /simul_20180523133828/proceso_16780/prueba.dat
Proceso 4: Completadas 50 escrituras en /simul_20180523133828/proceso_16782/prueba.dat
Proceso 5: Completadas 50 escrituras en /simul_20180523133828/proceso_16783/prueba.dat
Proceso 6: Completadas 50 escrituras en /simul_20180523133828/proceso_16784/prueba.dat
Proceso 7: Completadas 50 escrituras en /simul_20180523133828/proceso_16787/prueba.dat
Proceso 8: Completadas 50 escrituras en /simul_20180523133828/proceso_16788/prueba.dat
Proceso 9: Completadas 50 escrituras en /simul_20180523133828/proceso_16790/prueba.dat
```

[13] Time muestra 3 tiempos:

- el **real** de ejecución: es el tiempo total transcurrido desde que ha invocado el comando (incluyendo intervalos de tiempo utilizados por otros procesos y tiempos de espera para E/S). Se le denomina a veces como tiempo de reloj, porque es tiempo que ha transcurrido en nuestro reloj.
- el de **usuario:** es sólo el tiempo de CPU utilizado en la ejecución del proceso.
- el de **sistema:** es la cantidad de tiempo consumido en el kernel, que es el tiempo empleado en contestar peticiones del sistema.

```
Proceso 10: Completadas 50 escrituras en /simul_20180523133828/proceso_16791/prueba.dat
Proceso 11: Completadas 50 escrituras en /simul_20180523133828/proceso_16792/prueba.dat
Proceso 12: Completadas 50 escrituras en /simul_20180523133828/proceso_16795/prueba.dat
Proceso 13: Completadas 50 escrituras en /simul_20180523133828/proceso_16796/prueba.dat
Proceso 14: Completadas 50 escrituras en /simul_20180523133828/proceso_16797/prueba.dat
Proceso 16: Completadas 50 escrituras en /simul_20180523133828/proceso_16800/prueba.dat
Proceso 15: Completadas 50 escrituras en /simul_20180523133828/proceso_16799/prueba.dat
Proceso 17: Completadas 50 escrituras en /simul_20180523133828/proceso_16801/prueba.dat
Proceso 18: Completadas 50 escrituras en /simul_20180523133828/proceso_16803/prueba.dat
Proceso 19: Completadas 50 escrituras en /simul_20180523133828/proceso_16805/prueba.dat
Proceso 20: Completadas 50 escrituras en /simul_20180523133828/proceso_16806/prueba.dat
Proceso 21: Completadas 50 escrituras en /simul_20180523133828/proceso_16807/prueba.dat
Proceso 22: Completadas 50 escrituras en /simul_20180523133828/proceso_16810/prueba.dat
Proceso 23: Completadas 50 escrituras en /simul_20180523133828/proceso_16811/prueba.dat
Proceso 24: Completadas 50 escrituras en /simul_20180523133828/proceso_16814/prueba.dat
Proceso 25: Completadas 50 escrituras en /simul_20180523133828/proceso_16816/prueba.dat
Proceso 26: Completadas 50 escrituras en /simul_20180523133828/proceso_16817/prueba.dat
Proceso 27: Completadas 50 escrituras en /simul_20180523133828/proceso_16818/prueba.dat
Proceso 29: Completadas 50 escrituras en /simul_20180523133828/proceso_16821/prueba.dat
Proceso 28: Completadas 50 escrituras en /simul_20180523133828/proceso_16820/prueba.dat
Proceso 31: Completadas 50 escrituras en /simul_20180523133828/proceso_16823/prueba.dat
Proceso 30: Completadas 50 escrituras en /simul_20180523133828/proceso_16822/prueba.dat
Proceso 32: Completadas 50 escrituras en /simul_20180523133828/proceso_16824/prueba.dat
Proceso 33: Completadas 50 escrituras en /simul_20180523133828/proceso_16827/prueba.dat
Proceso 34: Completadas 50 escrituras en /simul_20180523133828/proceso_16828/prueba.dat
Proceso 35: Completadas 50 escrituras en /simul_20180523133828/proceso_16829/prueba.dat
Proceso 36: Completadas 50 escrituras en /simul_20180523133828/proceso_16831/prueba.dat
Proceso 37: Completadas 50 escrituras en /simul_20180523133828/proceso_16832/prueba.dat
Proceso 38: Completadas 50 escrituras en /simul_20180523133828/proceso_16833/prueba.dat
Proceso 39: Completadas 50 escrituras en /simul_20180523133828/proceso_16834/prueba.dat
Proceso 40: Completadas 50 escrituras en /simul_20180523133828/proceso_16836/prueba.dat
Proceso 41: Completadas 50 escrituras en /simul_20180523133828/proceso_16839/prueba.dat
Proceso 42: Completadas 50 escrituras en /simul_20180523133828/proceso_16840/prueba.dat
Proceso 44: Completadas 50 escrituras en /simul_20180523133828/proceso_16843/prueba.dat
Proceso 43: Completadas 50 escrituras en /simul_20180523133828/proceso_16842/prueba.dat
Proceso 45: Completadas 50 escrituras en /simul_20180523133828/proceso_16844/prueba.dat
Proceso 46: Completadas 50 escrituras en /simul_20180523133828/proceso_16845/prueba.dat
Proceso 47: Completadas 50 escrituras en /simul_20180523133828/proceso_16846/prueba.dat
Proceso 48: Completadas 50 escrituras en /simul_20180523133828/proceso_16847/prueba.dat
Proceso 50: Completadas 50 escrituras en /simul_20180523133828/proceso_16850/prueba.dat
Proceso 49: Completadas 50 escrituras en /simul_20180523133828/proceso_16849/prueba.dat
Proceso 52: Completadas 50 escrituras en /simul_20180523133828/proceso_16852/prueba.dat
Proceso 51: Completadas 50 escrituras en /simul_20180523133828/proceso_16851/prueba.dat
Proceso 53: Completadas 50 escrituras en /simul_20180523133828/proceso_16853/prueba.dat
Proceso 55: Completadas 50 escrituras en /simul_20180523133828/proceso_16856/prueba.dat
Proceso 56: Completadas 50 escrituras en /simul_20180523133828/proceso_16857/prueba.dat
Proceso 54: Completadas 50 escrituras en /simul_20180523133828/proceso_16855/prueba.dat
Proceso 57: Completadas 50 escrituras en /simul_20180523133828/proceso_16858/prueba.dat
```

```
Proceso 58: Completadas 50 escrituras en /simul_20180523133828/proceso_16860/prueba.dat
Proceso 59: Completadas 50 escrituras en /simul_20180523133828/proceso_16862/prueba.dat
Proceso 60: Completadas 50 escrituras en /simul_20180523133828/proceso_16863/prueba.dat
Proceso 61: Completadas 50 escrituras en /simul_20180523133828/proceso_16864/prueba.dat
Proceso 62: Completadas 50 escrituras en /simul_20180523133828/proceso_16865/prueba.dat
Proceso 63: Completadas 50 escrituras en /simul_20180523133828/proceso_16867/prueba.dat
Proceso 64: Completadas 50 escrituras en /simul_20180523133828/proceso_16868/prueba.dat
Proceso 65: Completadas 50 escrituras en /simul_20180523133828/proceso_16870/prueba.dat
Proceso 66: Completadas 50 escrituras en /simul_20180523133828/proceso_16872/prueba.dat
Proceso 67: Completadas 50 escrituras en /simul_20180523133828/proceso_16873/prueba.dat
Proceso 68: Completadas 50 escrituras en /simul_20180523133828/proceso_16874/prueba.dat
Proceso 69: Completadas 50 escrituras en /simul_20180523133828/proceso_16875/prueba.dat
Proceso 70: Completadas 50 escrituras en /simul_20180523133828/proceso_16876/prueba.dat
Proceso 71: Completadas 50 escrituras en /simul_20180523133828/proceso_16877/prueba.dat
Proceso 72: Completadas 50 escrituras en /simul_20180523133828/proceso_16879/prueba.dat
Proceso 73: Completadas 50 escrituras en /simul_20180523133828/proceso_16880/prueba.dat
Proceso 74: Completadas 50 escrituras en /simul_20180523133828/proceso_16881/prueba.dat
Proceso 75: Completadas 50 escrituras en /simul_20180523133828/proceso_16882/prueba.dat
Proceso 76: Completadas 50 escrituras en /simul_20180523133828/proceso_16883/prueba.dat
Proceso 77: Completadas 50 escrituras en /simul_20180523133828/proceso_16885/prueba.dat
Proceso 80: Completadas 50 escrituras en /simul_20180523133828/proceso_16888/prueba.dat
Proceso 78: Completadas 50 escrituras en /simul_20180523133828/proceso_16886/prueba.dat
Proceso 79: Completadas 50 escrituras en /simul_20180523133828/proceso_16887/prueba.dat
Proceso 81: Completadas 50 escrituras en /simul_20180523133828/proceso_16889/prueba.dat
Proceso 82: Completadas 50 escrituras en /simul_20180523133828/proceso_16890/prueba.dat
Proceso 83: Completadas 50 escrituras en /simul_20180523133828/proceso_16892/prueba.dat
Proceso 84: Completadas 50 escrituras en /simul_20180523133828/proceso_16893/prueba.dat
Proceso 85: Completadas 50 escrituras en /simul_20180523133828/proceso_16894/prueba.dat
Proceso 86: Completadas 50 escrituras en /simul_20180523133828/proceso_16895/prueba.dat
Proceso 88: Completadas 50 escrituras en /simul_20180523133828/proceso_16897/prueba.dat
Proceso 87: Completadas 50 escrituras en /simul_20180523133828/proceso_16896/prueba.dat
Proceso 89: Completadas 50 escrituras en /simul_20180523133828/proceso_16899/prueba.dat
Proceso 90: Completadas 50 escrituras en /simul_20180523133828/proceso_16901/prueba.dat
Proceso 91: Completadas 50 escrituras en /simul_20180523133828/proceso_16902/prueba.dat
Proceso 92: Completadas 50 escrituras en /simul_20180523133828/proceso_16904/prueba.dat
Proceso 93: Completadas 50 escrituras en /simul_20180523133828/proceso_16905/prueba.dat
Proceso 94: Completadas 50 escrituras en /simul_20180523133828/proceso_16906/prueba.dat
Proceso 95: Completadas 50 escrituras en /simul_20180523133828/proceso_16907/prueba.dat
Proceso 96: Completadas 50 escrituras en /simul_20180523133828/proceso_16908/prueba.dat
Proceso 97: Completadas 50 escrituras en /simul_20180523133828/proceso_16910/prueba.dat
Proceso 99: Completadas 50 escrituras en /simul_20180523133828/proceso_16912/prueba.dat
Proceso 98: Completadas 50 escrituras en /simul_20180523133828/proceso_16911/prueba.dat
Proceso 100: Completadas 50 escrituras en /simul_20180523133828/proceso_16914/prueba.dat
Total de procesos terminados: 100.

real    0m14.197s
user    0m0.239s
sys     0m0.745s
```

Observación: Un proceso puede acabar antes que otro iniciado antes que él

ANEXO 1

Makefile para compilar con la librería pthread:

```
CC=gcc
CFLAGS=-c -g -Wall -std=gnu99
LDFLAGS=-pthread

SOURCES=bloques.c ficheros_basico.c ficheros.c directorios.c mi_mkfs.c
leer_sf.c escribir.c leer.c truncar.c permitir.c mi_mkdir.c mi_chmod.c mi_ls.c
mi_link.c mi_escribir.c mi_cat.c mi_stat.c mi_rm.c semaforo_mutex_posix.c
simulacion.c #verificacion.c
LIBRARIES=bloques.o ficheros_basico.o ficheros.o directorios.o
semaforo_mutex_posix.o
INCLUDES=bloques.h ficheros_basico.h ficheros.h directorios.h
semaforo_mutex_posix.h simulacion.h #verificacion.h
PROGRAMS=mi_mkfs leer_sf escribir leer truncar permitir mi_mkdir mi_chmod
mi_ls mi_link mi_escribir mi_cat mi_stat mi_rm simulacion #verificacion
OBJS=$(SOURCES:.c=.o)

all: $(OBJS) $(PROGRAMS)

$(PROGRAMS): $(LIBRARIES) $(INCLUDES)
        $(CC) $(LDFLAGS) $(LIBRARIES) $@.o -o $@

%.o: %.c $(INCLUDES)
        $(CC) $(CFLAGS) -o $@ -c $<

.PHONY: clean
clean:
        rm -rf *.o *~ $(PROGRAMS)
```

Nivel 13: Verificación de las escrituras

En este nivel se creará un nuevo programa, verificacion.c, que recorrerá el fichero "`prueba.dat`" de cada proceso y generará la siguiente información:

- **Proceso**: escribir el PID
- **Número de escrituras**: escribir el contador de los registros *validados* dentro del fichero "`prueba.dat`" (se validan verificando que el campo PID coincida, ya que podríamos haber leído basura)
- **Primera escritura**: escribir la menor fecha y hora en formato epoch, el número de escritura en que ocurrió y su posición (nº registro)
- **Última escritura**: escribir la mayor fecha y hora en formato epoch, el número de escritura en que ocurrió y su posición (nº registro)
- **Menor posición**: escribir la posición (nº registro) más baja, el número de escritura en que ocurrió y su fecha y hora en formato epoch
- **Mayor posición**: escribir la posición (nº registro) más alta, el número de escritura en que ocurrió y su fecha y hora en formato epoch

El objetivo es imprimir por pantalla los cuatro registros más significativos que cada proceso ha escrito en su fichero.

```
//verificacion.h
struct INFORMACION {
   int pid;
   unsigned int nEscrituras; // validadas [1]
   struct REGISTRO PrimeraEscritura; [2]
   struct REGISTRO UltimaEscritura;
   struct REGISTRO MenorPosicion;
   struct REGISTRO MayorPosicion;
};
```

Algoritmo detallado:

Comprobar el nº de argumentos; // uso: ./verificacion <disco> **<directorio_simulacion>**
Montar disco.
Realizar el stat() del directorio de simulación para obtener su tamaño y calcular el nº de entradas.
Si el nº de entradas != NUMPROCESOS entonces devolver -1 fsi
Crear el fichero informe.txt dentro del directorio de simulación.

[1] El número de escrituras, por regla general, tiene que valer **50** (a menos que haya casualmente algún solapamiento).
[2] El tipo struct REGISTRO ya lo tenemos declarado en simulacion.h

Para cada entrada del directorio de simulación (es decir, para cada proceso) hacer
  Leer la entrada de directorio. [3]
  Extraer el PID a partir del nombre de la entrada y guardarlo en el registro info. [4]

  /* Recorrer **secuencialmente** el fichero prueba.dat utilizando un buffer de N registros de escrituras: [5] */
  Mientras haya escrituras en prueba.dat hacer [6]
    Leer una escritura
    Si la escritura es válida [7] entonces
      Si es la primera escritura validada entonces
        Inicializar los registros significativos con los datos de esa escritura (que ya será la de **menor posición**). [8]
      si_no
        Comparar fecha [9] y, para una misma fecha, el nº de escritura (para obtener la **primera** y la **última** escritura) [10] con los datos de nuestros registros y actualizar éstos si es necesario.
      fsi
      Incrementar el contador de escrituras validadas.
  fmientras
  Obtener la escritura de la **última posición** //será la delimitada por el EOF
  Añadir la información del struct info al fichero informe.txt por el final
fpara

[3] **Mejora**: Las entradas también las podéis haber leído todas de golpe previamente al inicio del bucle, con una sola llamada a mi_read() utilizando un buffer del tamaño NUMPROCESOS * sizeof (struct entrada) o llamando a vuestra función mi_dir() en su versión simple . Entonces en este paso lo que habría que hacer es leer la entrada pero del buffer, sin necesidad de acceder para cada una al dispositivo.

[4] Podéis utilizar la función strchr() con el carácter '_' para obtener los caracteres del PID, y luego pasarlos a entero con la función atoi(). El registro info es de tipo struct INFORMACION.

[5] La cantidad de registros, multiplicado por el sizeof(struct REGISTRO) mejor si es un múltiplo de BLOCKSIZE. En una plataforma de 64 bits el tamaño del struct REGISTRO es 24 bytes, entonces N podría ser por ejemplo 256 que multiplicado por 24 da 6144, que equivale a 6 bloques de 1024Bs.

```
int cant_registros_buffer_escrituras = 256; //
struct REGISTRO buffer_escrituras [cant_registros_buffer_escrituras];
while (mi_read(prueba,buffer_escrituras,offset, sizeof(buffer_escrituras)) > 0) { … }
```

Hay que limpiar el buffer de lectura antes de cada nuevo uso!!!

[6] Se ahorrará tiempo de ejecución teniendo una caché de directorios para realizar el mi_read() y no tener que llamar a buscar_entrada() para leer cada escritura de un mismo proceso (mismo camino). También si en vez de leer registro a registro del dispositivo, se explora en un buffer en memoria principal los registros que caben en un bloque, o en varios

[7] Para saberlo verificaremos que el campo pid de la escritura coincida con el del proceso, ya que podría haber basura debido a que tratamos con ficheros dispersos y hemos escrito en direcciones aleatorias

[8] info.PrimeraEscritura, info.UltimaEscritura, info.MenorPosicion, info.MayorPosicion

[9] Podéis utilizar la función difftime() para restar dos fechas. **Observación: No tenemos la suficiente precisión en segundos para discriminar sólo por fecha.**

[10] Dado que la precisión de la fecha no nos permite discriminar completamente cuál ha sido antes, entre las de fecha más temprana miraremos también el nº de escritura.

Desmontar disco.

Aunque las fechas se guarden en formato epoch se han de mostrar por pantalla en formato legible.

Por ejemplo con la función asctime():

```
asctime(localtime(&info.PrimeraEscritura.fecha))
```

se mostrará la fecha con el siguiente formato:

Fri May 25 11:44:47 2018

Y es más simple de utilizar, aunque menos flexible, que strftime(), que utilizábamos para ver los sellos de tiempo del inodo, y podíamos formatear a nuestro gusto. Por ejemplo usando el siguiente argumento `"%a %d-%m-%Y %H:%M:%S"` con strftime() se produciría el siguiente formato:

Fri 25-05-2018 11:44:47

Hay que mostrar por pantalla cuánto os tarda la verificación de escrituras. Podéis hacerlo desde consola mediante el comando time.

EJECUCIÓN DE LA VERIFICACIÓN DE LA SIMULACION DE EJEMPLO DE LA ETAPA ANTERIOR

```
$ /usr/bin/time ./verificacion disco /simul_20180523133828/
dir_sim: /simul_20180523133828/
numentradas: 100 NUMPROCESOS: 100
1) 50 escrituras validadas en /simul_20180523133828/proceso_16776/prueba.dat
2) 50 escrituras validadas en /simul_20180523133828/proceso_16779/prueba.dat
3) 50 escrituras validadas en /simul_20180523133828/proceso_16780/prueba.dat
4) 50 escrituras validadas en /simul_20180523133828/proceso_16782/prueba.dat
5) 50 escrituras validadas en /simul_20180523133828/proceso_16783/prueba.dat
6) 50 escrituras validadas en /simul_20180523133828/proceso_16784/prueba.dat
7) 50 escrituras validadas en /simul_20180523133828/proceso_16787/prueba.dat
8) 50 escrituras validadas en /simul_20180523133828/proceso_16788/prueba.dat
9) 50 escrituras validadas en /simul_20180523133828/proceso_16790/prueba.dat
10) 50 escrituras validadas en /simul_20180523133828/proceso_16791/prueba.dat
11) 50 escrituras validadas en /simul_20180523133828/proceso_16792/prueba.dat
12) 50 escrituras validadas en /simul_20180523133828/proceso_16795/prueba.dat
13) 50 escrituras validadas en /simul_20180523133828/proceso_16796/prueba.dat
14) 50 escrituras validadas en /simul_20180523133828/proceso_16797/prueba.dat
15) 50 escrituras validadas en /simul_20180523133828/proceso_16799/prueba.dat
16) 50 escrituras validadas en /simul_20180523133828/proceso_16800/prueba.dat
17) 50 escrituras validadas en /simul_20180523133828/proceso_16801/prueba.dat
18) 50 escrituras validadas en /simul_20180523133828/proceso_16803/prueba.dat
19) 50 escrituras validadas en /simul_20180523133828/proceso_16805/prueba.dat
20) 50 escrituras validadas en /simul_20180523133828/proceso_16806/prueba.dat
```

21) 50 escrituras validadas en /simul_20180523133828/proceso_16807/prueba.dat
22) 50 escrituras validadas en /simul_20180523133828/proceso_16810/prueba.dat
23) 50 escrituras validadas en /simul_20180523133828/proceso_16811/prueba.dat
24) 50 escrituras validadas en /simul_20180523133828/proceso_16814/prueba.dat
25) 50 escrituras validadas en /simul_20180523133828/proceso_16816/prueba.dat
26) 50 escrituras validadas en /simul_20180523133828/proceso_16817/prueba.dat
27) 50 escrituras validadas en /simul_20180523133828/proceso_16818/prueba.dat
28) 50 escrituras validadas en /simul_20180523133828/proceso_16820/prueba.dat
29) 50 escrituras validadas en /simul_20180523133828/proceso_16821/prueba.dat
30) 50 escrituras validadas en /simul_20180523133828/proceso_16822/prueba.dat
31) 50 escrituras validadas en /simul_20180523133828/proceso_16823/prueba.dat
32) 50 escrituras validadas en /simul_20180523133828/proceso_16824/prueba.dat
33) 50 escrituras validadas en /simul_20180523133828/proceso_16827/prueba.dat
34) 50 escrituras validadas en /simul_20180523133828/proceso_16828/prueba.dat
35) 50 escrituras validadas en /simul_20180523133828/proceso_16829/prueba.dat
36) 50 escrituras validadas en /simul_20180523133828/proceso_16831/prueba.dat
37) 50 escrituras validadas en /simul_20180523133828/proceso_16832/prueba.dat
38) 50 escrituras validadas en /simul_20180523133828/proceso_16833/prueba.dat
39) 50 escrituras validadas en /simul_20180523133828/proceso_16834/prueba.dat
40) 50 escrituras validadas en /simul_20180523133828/proceso_16836/prueba.dat
41) 50 escrituras validadas en /simul_20180523133828/proceso_16839/prueba.dat
42) 50 escrituras validadas en /simul_20180523133828/proceso_16840/prueba.dat
43) 50 escrituras validadas en /simul_20180523133828/proceso_16842/prueba.dat
44) 50 escrituras validadas en /simul_20180523133828/proceso_16843/prueba.dat
45) 50 escrituras validadas en /simul_20180523133828/proceso_16844/prueba.dat
46) 50 escrituras validadas en /simul_20180523133828/proceso_16845/prueba.dat
47) 50 escrituras validadas en /simul_20180523133828/proceso_16846/prueba.dat
48) 50 escrituras validadas en /simul_20180523133828/proceso_16847/prueba.dat
49) 50 escrituras validadas en /simul_20180523133828/proceso_16849/prueba.dat
50) 50 escrituras validadas en /simul_20180523133828/proceso_16850/prueba.dat
51) 50 escrituras validadas en /simul_20180523133828/proceso_16851/prueba.dat
52) 50 escrituras validadas en /simul_20180523133828/proceso_16852/prueba.dat
53) 50 escrituras validadas en /simul_20180523133828/proceso_16853/prueba.dat
54) 50 escrituras validadas en /simul_20180523133828/proceso_16855/prueba.dat
55) 50 escrituras validadas en /simul_20180523133828/proceso_16856/prueba.dat
56) 50 escrituras validadas en /simul_20180523133828/proceso_16857/prueba.dat
57) 50 escrituras validadas en /simul_20180523133828/proceso_16858/prueba.dat
58) 50 escrituras validadas en /simul_20180523133828/proceso_16860/prueba.dat
59) 50 escrituras validadas en /simul_20180523133828/proceso_16862/prueba.dat
60) 50 escrituras validadas en /simul_20180523133828/proceso_16863/prueba.dat
61) 50 escrituras validadas en /simul_20180523133828/proceso_16864/prueba.dat
62) 50 escrituras validadas en /simul_20180523133828/proceso_16865/prueba.dat
63) 50 escrituras validadas en /simul_20180523133828/proceso_16867/prueba.dat
64) 50 escrituras validadas en /simul_20180523133828/proceso_16868/prueba.dat
65) 50 escrituras validadas en /simul_20180523133828/proceso_16870/prueba.dat
66) 50 escrituras validadas en /simul_20180523133828/proceso_16872/prueba.dat
67) 50 escrituras validadas en /simul_20180523133828/proceso_16873/prueba.dat
68) 50 escrituras validadas en /simul_20180523133828/proceso_16874/prueba.dat

69) 50 escrituras validadas en /simul_20180523133828/proceso_16875/prueba.dat
70) 50 escrituras validadas en /simul_20180523133828/proceso_16876/prueba.dat
71) 50 escrituras validadas en /simul_20180523133828/proceso_16877/prueba.dat
72) 50 escrituras validadas en /simul_20180523133828/proceso_16879/prueba.dat
73) 50 escrituras validadas en /simul_20180523133828/proceso_16880/prueba.dat
74) 50 escrituras validadas en /simul_20180523133828/proceso_16881/prueba.dat
75) 50 escrituras validadas en /simul_20180523133828/proceso_16882/prueba.dat
76) 50 escrituras validadas en /simul_20180523133828/proceso_16883/prueba.dat
77) 50 escrituras validadas en /simul_20180523133828/proceso_16885/prueba.dat
78) 50 escrituras validadas en /simul_20180523133828/proceso_16886/prueba.dat
79) 50 escrituras validadas en /simul_20180523133828/proceso_16887/prueba.dat
80) 50 escrituras validadas en /simul_20180523133828/proceso_16888/prueba.dat
81) 50 escrituras validadas en /simul_20180523133828/proceso_16889/prueba.dat
82) 50 escrituras validadas en /simul_20180523133828/proceso_16890/prueba.dat
83) 50 escrituras validadas en /simul_20180523133828/proceso_16892/prueba.dat
84) 50 escrituras validadas en /simul_20180523133828/proceso_16893/prueba.dat
85) 50 escrituras validadas en /simul_20180523133828/proceso_16894/prueba.dat
86) 50 escrituras validadas en /simul_20180523133828/proceso_16895/prueba.dat
87) 50 escrituras validadas en /simul_20180523133828/proceso_16896/prueba.dat
88) 50 escrituras validadas en /simul_20180523133828/proceso_16897/prueba.dat
89) 50 escrituras validadas en /simul_20180523133828/proceso_16899/prueba.dat
90) 50 escrituras validadas en /simul_20180523133828/proceso_16901/prueba.dat
91) 50 escrituras validadas en /simul_20180523133828/proceso_16902/prueba.dat
92) 50 escrituras validadas en /simul_20180523133828/proceso_16904/prueba.dat
93) 50 escrituras validadas en /simul_20180523133828/proceso_16905/prueba.dat
94) 50 escrituras validadas en /simul_20180523133828/proceso_16906/prueba.dat
95) 50 escrituras validadas en /simul_20180523133828/proceso_16907/prueba.dat
96) 50 escrituras validadas en /simul_20180523133828/proceso_16908/prueba.dat
97) 50 escrituras validadas en /simul_20180523133828/proceso_16910/prueba.dat
98) 50 escrituras validadas en /simul_20180523133828/proceso_16911/prueba.dat
99) 50 escrituras validadas en /simul_20180523133828/proceso_16912/prueba.dat
100) 50 escrituras validadas en /simul_20180523133828/proceso_16914/prueba.dat

real 0m6.834s
user 0m1.797s
sys 0m5.015s

$ ./mi_cat disco /simul_20180523133828/informe.txt > resultado.txt

Total_leidos 25362 [11]

$ ls -l resultado.txt
-rw-r--r-- 1 uib uib 25362 may 23 14:08 resultado.txt

$ cat resultado.txt

[11] A cada uno os puede dar un valor diferente según el formato de vuestro informe, pero ha de coincidir con el valor del tamaño del fichero externo al hacer luego el ls -l

```
PID: 16776
Numero de escrituras: 50
Primera Escritura       1      327582  Wed May 23 13:38:28 2018
Ultima Escritura        50     360073  Wed May 23 13:38:30 2018
Menor Posición          22      1068    Wed May 23 13:38:29 2018
Mayor Posición          46     489912  Wed May 23 13:38:30 2018

PID: 16779
Numero de escrituras: 50
Primera Escritura       1      208455  Wed May 23 13:38:28 2018
Ultima Escritura        50     483004  Wed May 23 13:38:31 2018
Menor Posición          30      19396   Wed May 23 13:38:30 2018
Mayor Posición          39     498365  Wed May 23 13:38:30 2018

PID: 16780
Numero de escrituras: 50
Primera Escritura       1      278357  Wed May 23 13:38:28 2018
Ultima Escritura        50     420918  Wed May 23 13:38:31 2018
Menor Posición          23      12032   Wed May 23 13:38:30 2018
Mayor Posición          32     497756  Wed May 23 13:38:30 2018

PID: 16782
Numero de escrituras: 50
Primera Escritura       1      97475   Wed May 23 13:38:29 2018
Ultima Escritura        50     237833  Wed May 23 13:38:31 2018
Menor Posición          24      874     Wed May 23 13:38:30 2018
Mayor Posición          19     497300  Wed May 23 13:38:30 2018

PID: 16783
Numero de escrituras: 50
Primera Escritura       1      292033  Wed May 23 13:38:29 2018
Ultima Escritura        50     362953  Wed May 23 13:38:31 2018
Menor Posición          38      18509   Wed May 23 13:38:31 2018
Mayor Posición          14     484921  Wed May 23 13:38:29 2018

PID: 16784
Numero de escrituras: 50
Primera Escritura       1      491400  Wed May 23 13:38:29 2018
Ultima Escritura        50     55410   Wed May 23 13:38:31 2018
Menor Posición          18      15871   Wed May 23 13:38:30 2018
Mayor Posición          21     494636  Wed May 23 13:38:30 2018

PID: 16787
Numero de escrituras: 50
Primera Escritura       1      178428  Wed May 23 13:38:29 2018
Ultima Escritura        50     367966  Wed May 23 13:38:32 2018
Menor Posición          18      2242    Wed May 23 13:38:30 2018
```

```
Mayor Posición          30      493993  Wed May 23 13:38:31 2018

PID: 16788
Numero de escrituras: 50
Primera Escritura       1       148384  Wed May 23 13:38:29 2018
Ultima Escritura        50      302203  Wed May 23 13:38:32 2018
Menor Posición          42      9875    Wed May 23 13:38:31 2018
Mayor Posición          15      493693  Wed May 23 13:38:30 2018

PID: 16790
Numero de escrituras: 50
Primera Escritura       1       165649  Wed May 23 13:38:30 2018
Ultima Escritura        50      189012  Wed May 23 13:38:32 2018
Menor Posición          32      10359   Wed May 23 13:38:31 2018
Mayor Posición          42      497658  Wed May 23 13:38:32 2018

PID: 16791
Numero de escrituras: 50
Primera Escritura       1       420179  Wed May 23 13:38:30 2018
Ultima Escritura        50      68465   Wed May 23 13:38:32 2018
Menor Posición          26      10510   Wed May 23 13:38:31 2018
Mayor Posición          39      497360  Wed May 23 13:38:32 2018

PID: 16792
Numero de escrituras: 50
Primera Escritura       1       113026  Wed May 23 13:38:30 2018
Ultima Escritura        50      247864  Wed May 23 13:38:33 2018
Menor Posición          35      31379   Wed May 23 13:38:32 2018
Mayor Posición          43      485071  Wed May 23 13:38:32 2018

PID: 16795
Numero de escrituras: 50
Primera Escritura       1       170396  Wed May 23 13:38:30 2018
Ultima Escritura        50      163921  Wed May 23 13:38:33 2018
Menor Posición          39      2631    Wed May 23 13:38:32 2018
Mayor Posición          49      499721  Wed May 23 13:38:33 2018

PID: 16796
Numero de escrituras: 50
Primera Escritura       1       496888  Wed May 23 13:38:30 2018
Ultima Escritura        50      398611  Wed May 23 13:38:33 2018
Menor Posición          17      5770    Wed May 23 13:38:31 2018
Mayor Posición          1       496888  Wed May 23 13:38:30 2018

PID: 16797
Numero de escrituras: 50
Primera Escritura       1       159163  Wed May 23 13:38:31 2018
Ultima Escritura        50      469182  Wed May 23 13:38:33 2018
```

```
Menor Posición        31     21602   Wed May 23 13:38:32 2018
Mayor Posición        48     488696  Wed May 23 13:38:33 2018

PID: 16799
Numero de escrituras: 50
Primera Escritura     1      163046  Wed May 23 13:38:31 2018
Ultima Escritura      50     365331  Wed May 23 13:38:33 2018
Menor Posición        36     1031    Wed May 23 13:38:33 2018
Mayor Posición        43     495136  Wed May 23 13:38:33 2018

PID: 16800
Numero de escrituras: 50
Primera Escritura     1      166331  Wed May 23 13:38:31 2018
Ultima Escritura      50     115531  Wed May 23 13:38:33 2018
Menor Posición        17     3895    Wed May 23 13:38:32 2018
Mayor Posición        34     499204  Wed May 23 13:38:32 2018

PID: 16801
Numero de escrituras: 50
Primera Escritura     1      189620  Wed May 23 13:38:31 2018
Ultima Escritura      50     130280  Wed May 23 13:38:33 2018
Menor Posición        30     14238   Wed May 23 13:38:32 2018
Mayor Posición        44     488945  Wed May 23 13:38:33 2018

PID: 16803
Numero de escrituras: 50
Primera Escritura     1      100551  Wed May 23 13:38:31 2018
Ultima Escritura      50     352039  Wed May 23 13:38:34 2018
Menor Posición        16     3567    Wed May 23 13:38:32 2018
Mayor Posición        32     491102  Wed May 23 13:38:33 2018

PID: 16805
Numero de escrituras: 50
Primera Escritura     1      138800  Wed May 23 13:38:31 2018
Ultima Escritura      50     381488  Wed May 23 13:38:34 2018
Menor Posición        15     13180   Wed May 23 13:38:32 2018
Mayor Posición        30     496659  Wed May 23 13:38:33 2018

PID: 16806
Numero de escrituras: 50
Primera Escritura     1      132076  Wed May 23 13:38:32 2018
Ultima Escritura      50     294927  Wed May 23 13:38:34 2018
Menor Posición        14     20150   Wed May 23 13:38:32 2018
Mayor Posición        11     471852  Wed May 23 13:38:32 2018

PID: 16807
Numero de escrituras: 50
Primera Escritura     1      89484   Wed May 23 13:38:32 2018
```

```
Ultima Escritura       50     117297  Wed May 23 13:38:34 2018
Menor Posición         16      1188    Wed May 23 13:38:32 2018
Mayor Posición         42      495825  Wed May 23 13:38:34 2018

PID: 16810
Numero de escrituras: 50
Primera Escritura      1      486234  Wed May 23 13:38:32 2018
Ultima Escritura       50     217327  Wed May 23 13:38:34 2018
Menor Posición         20      13573   Wed May 23 13:38:33 2018
Mayor Posición         8      495166  Wed May 23 13:38:32 2018

PID: 16811
Numero de escrituras: 50
Primera Escritura      1      348681  Wed May 23 13:38:32 2018
Ultima Escritura       50     257205  Wed May 23 13:38:34 2018
Menor Posición         36      1345    Wed May 23 13:38:34 2018
Mayor Posición         37      495159  Wed May 23 13:38:34 2018

PID: 16814
Numero de escrituras: 50
Primera Escritura      1      399016  Wed May 23 13:38:32 2018
Ultima Escritura       50     158824  Wed May 23 13:38:35 2018
Menor Posición         41      34658   Wed May 23 13:38:34 2018
Mayor Posición         18      495583  Wed May 23 13:38:33 2018

PID: 16816
Numero de escrituras: 50
Primera Escritura      1      107469  Wed May 23 13:38:32 2018
Ultima Escritura       50     388917  Wed May 23 13:38:35 2018
Menor Posición         37      38995   Wed May 23 13:38:34 2018
Mayor Posición         42      484872  Wed May 23 13:38:34 2018

PID: 16817
Numero de escrituras: 50
Primera Escritura      1      179477  Wed May 23 13:38:33 2018
Ultima Escritura       50     145852  Wed May 23 13:38:35 2018
Menor Posición         41      7782    Wed May 23 13:38:35 2018
Mayor Posición         48      487236  Wed May 23 13:38:35 2018

PID: 16818
Numero de escrituras: 50
Primera Escritura      1      81585   Wed May 23 13:38:33 2018
Ultima Escritura       50     161100  Wed May 23 13:38:35 2018
Menor Posición         4      768     Wed May 23 13:38:33 2018
Mayor Posición         41      490458  Wed May 23 13:38:35 2018

PID: 16820
Numero de escrituras: 50
```

```
Primera Escritura       1       353501  Wed May 23 13:38:33 2018
Ultima Escritura        50      341456  Wed May 23 13:38:35 2018
Menor Posición          25      5966    Wed May 23 13:38:34 2018
Mayor Posición          11      496892  Wed May 23 13:38:33 2018

PID: 16821
Numero de escrituras: 50
Primera Escritura       1       449296  Wed May 23 13:38:33 2018
Ultima Escritura        50      428843  Wed May 23 13:38:35 2018
Menor Posición          19      2897    Wed May 23 13:38:34 2018
Mayor Posición          7       494374  Wed May 23 13:38:33 2018

PID: 16822
Numero de escrituras: 50
Primera Escritura       1       395939  Wed May 23 13:38:33 2018
Ultima Escritura        50      269544  Wed May 23 13:38:35 2018
Menor Posición          7       28562   Wed May 23 13:38:33 2018
Mayor Posición          24      479848  Wed May 23 13:38:34 2018

PID: 16823
Numero de escrituras: 50
Primera Escritura       1       117415  Wed May 23 13:38:33 2018
Ultima Escritura        50      304915  Wed May 23 13:38:35 2018
Menor Posición          14      2259    Wed May 23 13:38:34 2018
Mayor Posición          26      499817  Wed May 23 13:38:34 2018

PID: 16824
Numero de escrituras: 50
Primera Escritura       1       390950  Wed May 23 13:38:33 2018
Ultima Escritura        50      272263  Wed May 23 13:38:36 2018
Menor Posición          17      9071    Wed May 23 13:38:34 2018
Mayor Posición          19      491680  Wed May 23 13:38:34 2018

PID: 16827
Numero de escrituras: 50
Primera Escritura       1       319750  Wed May 23 13:38:33 2018
Ultima Escritura        50      313528  Wed May 23 13:38:36 2018
Menor Posición          18      1152    Wed May 23 13:38:34 2018
Mayor Posición          33      487039  Wed May 23 13:38:35 2018

PID: 16828
Numero de escrituras: 50
Primera Escritura       1       282422  Wed May 23 13:38:33 2018
Ultima Escritura        50      227780  Wed May 23 13:38:36 2018
Menor Posición          8       11709   Wed May 23 13:38:34 2018
Mayor Posición          2       493673  Wed May 23 13:38:33 2018

PID: 16829
```

```
Numero de escrituras: 50
Primera Escritura       1      26503   Wed May 23 13:38:33 2018
Ultima Escritura        50     296169  Wed May 23 13:38:36 2018
Menor Posición          24      3328    Wed May 23 13:38:35 2018
Mayor Posición          41      485111  Wed May 23 13:38:35 2018

PID: 16831
Numero de escrituras: 50
Primera Escritura       1      190107  Wed May 23 13:38:34 2018
Ultima Escritura        50     188145  Wed May 23 13:38:36 2018
Menor Posición          15      12279   Wed May 23 13:38:34 2018
Mayor Posición          12      496019  Wed May 23 13:38:34 2018

PID: 16832
Numero de escrituras: 50
Primera Escritura       1      202010  Wed May 23 13:38:34 2018
Ultima Escritura        50     156090  Wed May 23 13:38:36 2018
Menor Posición          41      8316    Wed May 23 13:38:36 2018
Mayor Posición          30      499254  Wed May 23 13:38:35 2018

PID: 16833
Numero de escrituras: 50
Primera Escritura       1      287262  Wed May 23 13:38:34 2018
Ultima Escritura        50     90195   Wed May 23 13:38:36 2018
Menor Posición          29      2338    Wed May 23 13:38:35 2018
Mayor Posición          23      490353  Wed May 23 13:38:35 2018

PID: 16834
Numero de escrituras: 50
Primera Escritura       1      219757  Wed May 23 13:38:34 2018
Ultima Escritura        50     452249  Wed May 23 13:38:37 2018
Menor Posición          40      1244    Wed May 23 13:38:36 2018
Mayor Posición          45      496883  Wed May 23 13:38:36 2018

PID: 16836
Numero de escrituras: 50
Primera Escritura       1      69452   Wed May 23 13:38:34 2018
Ultima Escritura        50     253207  Wed May 23 13:38:37 2018
Menor Posición          49      3292    Wed May 23 13:38:36 2018
Mayor Posición          45      497375  Wed May 23 13:38:36 2018

PID: 16839
Numero de escrituras: 50
Primera Escritura       1      212920  Wed May 23 13:38:34 2018
Ultima Escritura        50     140821  Wed May 23 13:38:37 2018
Menor Posición          33      5764    Wed May 23 13:38:36 2018
Mayor Posición          19      492660  Wed May 23 13:38:35 2018
```

```
PID: 16840
Numero de escrituras: 50
Primera Escritura       1       129209  Wed May 23 13:38:34 2018
Ultima Escritura        50      204891  Wed May 23 13:38:37 2018
Menor Posición          7       16745   Wed May 23 13:38:35 2018
Mayor Posición          24      494242  Wed May 23 13:38:36 2018

PID: 16842
Numero de escrituras: 50
Primera Escritura       1       82406   Wed May 23 13:38:35 2018
Ultima Escritura        50      496944  Wed May 23 13:38:37 2018
Menor Posición          21      5658    Wed May 23 13:38:36 2018
Mayor Posición          50      496944  Wed May 23 13:38:37 2018

PID: 16843
Numero de escrituras: 50
Primera Escritura       1       142365  Wed May 23 13:38:35 2018
Ultima Escritura        50      90700   Wed May 23 13:38:37 2018
Menor Posición          6       15225   Wed May 23 13:38:35 2018
Mayor Posición          26      489190  Wed May 23 13:38:36 2018

PID: 16844
Numero de escrituras: 50
Primera Escritura       1       365208  Wed May 23 13:38:35 2018
Ultima Escritura        50      344950  Wed May 23 13:38:37 2018
Menor Posición          31      5070    Wed May 23 13:38:36 2018
Mayor Posición          4       484207  Wed May 23 13:38:35 2018

PID: 16845
Numero de escrituras: 50
Primera Escritura       1       106001  Wed May 23 13:38:35 2018
Ultima Escritura        50      499450  Wed May 23 13:38:37 2018
Menor Posición          13      6901    Wed May 23 13:38:35 2018
Mayor Posición          50      499450  Wed May 23 13:38:37 2018

PID: 16846
Numero de escrituras: 50
Primera Escritura       1       209540  Wed May 23 13:38:35 2018
Ultima Escritura        50      96486   Wed May 23 13:38:38 2018
Menor Posición          2       51547   Wed May 23 13:38:35 2018
Mayor Posición          11      495087  Wed May 23 13:38:36 2018

PID: 16847
Numero de escrituras: 50
Primera Escritura       1       71611   Wed May 23 13:38:35 2018
Ultima Escritura        50      472623  Wed May 23 13:38:38 2018
Menor Posición          24      3227    Wed May 23 13:38:36 2018
Mayor Posición          3       482274  Wed May 23 13:38:35 2018
```

```
PID: 16849
Numero de escrituras: 50
Primera Escritura       1      42709   Wed May 23 13:38:35 2018
Ultima Escritura       50      45089   Wed May 23 13:38:38 2018
Menor Posición         26       5030   Wed May 23 13:38:37 2018
Mayor Posición          8     499639   Wed May 23 13:38:36 2018

PID: 16850
Numero de escrituras: 50
Primera Escritura       1     449659   Wed May 23 13:38:35 2018
Ultima Escritura       50     298748   Wed May 23 13:38:38 2018
Menor Posición         10       9056   Wed May 23 13:38:36 2018
Mayor Posición         29     494783   Wed May 23 13:38:37 2018

PID: 16851
Numero de escrituras: 50
Primera Escritura       1      75702   Wed May 23 13:38:35 2018
Ultima Escritura       50      20304   Wed May 23 13:38:38 2018
Menor Posición         22      16781   Wed May 23 13:38:37 2018
Mayor Posición         48     484269   Wed May 23 13:38:38 2018

PID: 16852
Numero de escrituras: 50
Primera Escritura       1     184917   Wed May 23 13:38:35 2018
Ultima Escritura       50     246034   Wed May 23 13:38:38 2018
Menor Posición         14      28324   Wed May 23 13:38:36 2018
Mayor Posición         19     497067   Wed May 23 13:38:36 2018

PID: 16853
Numero de escrituras: 50
Primera Escritura       1       8685   Wed May 23 13:38:36 2018
Ultima Escritura       50     363856   Wed May 23 13:38:38 2018
Menor Posición          1       8685   Wed May 23 13:38:36 2018
Mayor Posición         32     479681   Wed May 23 13:38:37 2018

PID: 16855
Numero de escrituras: 50
Primera Escritura       1     282993   Wed May 23 13:38:36 2018
Ultima Escritura       50     140565   Wed May 23 13:38:38 2018
Menor Posición          7       2591   Wed May 23 13:38:36 2018
Mayor Posición          9     484453   Wed May 23 13:38:36 2018

PID: 16856
Numero de escrituras: 50
Primera Escritura       1     397724   Wed May 23 13:38:36 2018
Ultima Escritura       50     204661   Wed May 23 13:38:38 2018
Menor Posición          7       1372   Wed May 23 13:38:36 2018
```

```
Mayor Posición          34      490853  Wed May 23 13:38:37 2018

PID: 16857
Numero de escrituras: 50
Primera Escritura       1       455409  Wed May 23 13:38:36 2018
Ultima Escritura        50      397592  Wed May 23 13:38:38 2018
Menor Posición          33      2262    Wed May 23 13:38:37 2018
Mayor Posición          28      478031  Wed May 23 13:38:37 2018

PID: 16858
Numero de escrituras: 50
Primera Escritura       1       102793  Wed May 23 13:38:36 2018
Ultima Escritura        50      432289  Wed May 23 13:38:38 2018
Menor Posición          49      3302    Wed May 23 13:38:38 2018
Mayor Posición          7       491479  Wed May 23 13:38:36 2018

PID: 16860
Numero de escrituras: 50
Primera Escritura       1       129430  Wed May 23 13:38:36 2018
Ultima Escritura        50      176014  Wed May 23 13:38:39 2018
Menor Posición          39      9719    Wed May 23 13:38:38 2018
Mayor Posición          6       499994  Wed May 23 13:38:36 2018

PID: 16862
Numero de escrituras: 50
Primera Escritura       1       373647  Wed May 23 13:38:36 2018
Ultima Escritura        50      214344  Wed May 23 13:38:39 2018
Menor Posición          5       2506    Wed May 23 13:38:36 2018
Mayor Posición          40      481124  Wed May 23 13:38:38 2018

PID: 16863
Numero de escrituras: 50
Primera Escritura       1       142489  Wed May 23 13:38:36 2018
Ultima Escritura        50      208699  Wed May 23 13:38:39 2018
Menor Posición          21      103     Wed May 23 13:38:37 2018
Mayor Posición          4       490331  Wed May 23 13:38:37 2018

PID: 16864
Numero de escrituras: 50
Primera Escritura       1       175296  Wed May 23 13:38:37 2018
Ultima Escritura        50      221416  Wed May 23 13:38:39 2018
Menor Posición          48      27192   Wed May 23 13:38:39 2018
Mayor Posición          36      483603  Wed May 23 13:38:38 2018

PID: 16865
Numero de escrituras: 50
Primera Escritura       1       363638  Wed May 23 13:38:37 2018
Ultima Escritura        50      152445  Wed May 23 13:38:39 2018
```

```
Menor Posición        17     6001    Wed May 23 13:38:37 2018
Mayor Posición        43     492434  Wed May 23 13:38:39 2018

PID: 16867
Numero de escrituras: 50
Primera Escritura     1      280619  Wed May 23 13:38:37 2018
Ultima Escritura      50     12992   Wed May 23 13:38:39 2018
Menor Posición        50     12992   Wed May 23 13:38:39 2018
Mayor Posición        24     499517  Wed May 23 13:38:38 2018

PID: 16868
Numero de escrituras: 50
Primera Escritura     1      103659  Wed May 23 13:38:37 2018
Ultima Escritura      50     449454  Wed May 23 13:38:39 2018
Menor Posición        22     676     Wed May 23 13:38:38 2018
Mayor Posición        21     498752  Wed May 23 13:38:38 2018

PID: 16870
Numero de escrituras: 50
Primera Escritura     1      73508   Wed May 23 13:38:37 2018
Ultima Escritura      50     396433  Wed May 23 13:38:40 2018
Menor Posición        27     4478    Wed May 23 13:38:38 2018
Mayor Posición        43     487846  Wed May 23 13:38:39 2018

PID: 16872
Numero de escrituras: 50
Primera Escritura     1      409746  Wed May 23 13:38:37 2018
Ultima Escritura      50     259329  Wed May 23 13:38:40 2018
Menor Posición        46     10274   Wed May 23 13:38:39 2018
Mayor Posición        5      484422  Wed May 23 13:38:37 2018

PID: 16873
Numero de escrituras: 50
Primera Escritura     1      156029  Wed May 23 13:38:37 2018
Ultima Escritura      50     314741  Wed May 23 13:38:40 2018
Menor Posición        40     45461   Wed May 23 13:38:39 2018
Mayor Posición        19     498194  Wed May 23 13:38:38 2018

PID: 16874
Numero de escrituras: 50
Primera Escritura     1      29782   Wed May 23 13:38:37 2018
Ultima Escritura      50     460440  Wed May 23 13:38:40 2018
Menor Posición        45     5068    Wed May 23 13:38:40 2018
Mayor Posición        13     499628  Wed May 23 13:38:38 2018

PID: 16875
Numero de escrituras: 50
Primera Escritura     1      251961  Wed May 23 13:38:37 2018
```

```
Ultima Escritura       50    405192  Wed May 23 13:38:40 2018
Menor Posición         36     237    Wed May 23 13:38:39 2018
Mayor Posición         35     492735  Wed May 23 13:38:39 2018

PID: 16876
Numero de escrituras: 50
Primera Escritura      1     410424  Wed May 23 13:38:38 2018
Ultima Escritura       50    21816   Wed May 23 13:38:40 2018
Menor Posición         36     13871   Wed May 23 13:38:39 2018
Mayor Posición         29     493634  Wed May 23 13:38:39 2018

PID: 16877
Numero de escrituras: 50
Primera Escritura      1     186061  Wed May 23 13:38:38 2018
Ultima Escritura       50    216997  Wed May 23 13:38:40 2018
Menor Posición         33     5282    Wed May 23 13:38:39 2018
Mayor Posición         15     475625  Wed May 23 13:38:38 2018

PID: 16879
Numero de escrituras: 50
Primera Escritura      1     276191  Wed May 23 13:38:38 2018
Ultima Escritura       50    174170  Wed May 23 13:38:40 2018
Menor Posición         42     6531    Wed May 23 13:38:40 2018
Mayor Posición         23     498261  Wed May 23 13:38:39 2018

PID: 16880
Numero de escrituras: 50
Primera Escritura      1     464375  Wed May 23 13:38:38 2018
Ultima Escritura       50    247054  Wed May 23 13:38:40 2018
Menor Posición         32     2955    Wed May 23 13:38:39 2018
Mayor Posición         19     485158  Wed May 23 13:38:39 2018

PID: 16881
Numero de escrituras: 50
Primera Escritura      1     171856  Wed May 23 13:38:38 2018
Ultima Escritura       50    281087  Wed May 23 13:38:40 2018
Menor Posición         10     22859   Wed May 23 13:38:38 2018
Mayor Posición         23     490046  Wed May 23 13:38:39 2018

PID: 16882
Numero de escrituras: 50
Primera Escritura      1     314801  Wed May 23 13:38:38 2018
Ultima Escritura       50    56673   Wed May 23 13:38:40 2018
Menor Posición         11     19074   Wed May 23 13:38:39 2018
Mayor Posición         15     490248  Wed May 23 13:38:39 2018

PID: 16883
Numero de escrituras: 50
```

```
Primera Escritura       1       253680  Wed May 23 13:38:38 2018
Ultima Escritura        50      473678  Wed May 23 13:38:40 2018
Menor Posición          37      32918   Wed May 23 13:38:40 2018
Mayor Posición          41      493591  Wed May 23 13:38:40 2018

PID: 16885
Numero de escrituras: 50
Primera Escritura       1       205818  Wed May 23 13:38:38 2018
Ultima Escritura        50      15097   Wed May 23 13:38:41 2018
Menor Posición          13      3324    Wed May 23 13:38:39 2018
Mayor Posición          17      485392  Wed May 23 13:38:39 2018

PID: 16886
Numero de escrituras: 50
Primera Escritura       1       113432  Wed May 23 13:38:38 2018
Ultima Escritura        50      209619  Wed May 23 13:38:41 2018
Menor Posición          37      20958   Wed May 23 13:38:40 2018
Mayor Posición          10      484486  Wed May 23 13:38:39 2018

PID: 16887
Numero de escrituras: 50
Primera Escritura       1       131169  Wed May 23 13:38:38 2018
Ultima Escritura        50      407211  Wed May 23 13:38:41 2018
Menor Posición          49      4550    Wed May 23 13:38:41 2018
Mayor Posición          11      456273  Wed May 23 13:38:39 2018

PID: 16888
Numero de escrituras: 50
Primera Escritura       1       347018  Wed May 23 13:38:38 2018
Ultima Escritura        50      242085  Wed May 23 13:38:41 2018
Menor Posición          43      5161    Wed May 23 13:38:40 2018
Mayor Posición          21      497195  Wed May 23 13:38:39 2018

PID: 16889
Numero de escrituras: 50
Primera Escritura       1       429469  Wed May 23 13:38:38 2018
Ultima Escritura        50      483965  Wed May 23 13:38:41 2018
Menor Posición          8       6013    Wed May 23 13:38:39 2018
Mayor Posición          50      483965  Wed May 23 13:38:41 2018

PID: 16890
Numero de escrituras: 50
Primera Escritura       1       238278  Wed May 23 13:38:39 2018
Ultima Escritura        50      183431  Wed May 23 13:38:41 2018
Menor Posición          25      506     Wed May 23 13:38:40 2018
Mayor Posición          36      498118  Wed May 23 13:38:40 2018

PID: 16892
```

```
Numero de escrituras: 50
Primera Escritura       1      257689  Wed May 23 13:38:39 2018
Ultima Escritura        50     344860  Wed May 23 13:38:41 2018
Menor Posición          11      36354   Wed May 23 13:38:39 2018
Mayor Posición          21     497441  Wed May 23 13:38:40 2018

PID: 16893
Numero de escrituras: 50
Primera Escritura       1      198108  Wed May 23 13:38:39 2018
Ultima Escritura        50     117206  Wed May 23 13:38:41 2018
Menor Posición          31      13733   Wed May 23 13:38:40 2018
Mayor Posición          35     494516  Wed May 23 13:38:40 2018

PID: 16894
Numero de escrituras: 50
Primera Escritura       1      191353  Wed May 23 13:38:39 2018
Ultima Escritura        50     179612  Wed May 23 13:38:41 2018
Menor Posición          41      3355    Wed May 23 13:38:41 2018
Mayor Posición          31     478580  Wed May 23 13:38:40 2018

PID: 16895
Numero de escrituras: 50
Primera Escritura       1      201914  Wed May 23 13:38:39 2018
Ultima Escritura        50     20460   Wed May 23 13:38:41 2018
Menor Posición          19      16541   Wed May 23 13:38:40 2018
Mayor Posición          47     487567  Wed May 23 13:38:41 2018

PID: 16896
Numero de escrituras: 50
Primera Escritura       1      392081  Wed May 23 13:38:39 2018
Ultima Escritura        50     379518  Wed May 23 13:38:42 2018
Menor Posición          19      414     Wed May 23 13:38:40 2018
Mayor Posición          13     493726  Wed May 23 13:38:40 2018

PID: 16897
Numero de escrituras: 50
Primera Escritura       1      236306  Wed May 23 13:38:39 2018
Ultima Escritura        50     332406  Wed May 23 13:38:42 2018
Menor Posición          21      10713   Wed May 23 13:38:40 2018
Mayor Posición          39     496532  Wed May 23 13:38:41 2018

PID: 16899
Numero de escrituras: 50
Primera Escritura       1      451340  Wed May 23 13:38:39 2018
Ultima Escritura        50     351768  Wed May 23 13:38:42 2018
Menor Posición          32      18273   Wed May 23 13:38:41 2018
Mayor Posición          40     498208  Wed May 23 13:38:41 2018
```

```
PID: 16901
Numero de escrituras: 50
Primera Escritura       1      70736   Wed May 23 13:38:39 2018
Ultima Escritura        50     20459   Wed May 23 13:38:42 2018
Menor Posición          15     561     Wed May 23 13:38:40 2018
Mayor Posición          32     485336  Wed May 23 13:38:41 2018

PID: 16902
Numero de escrituras: 50
Primera Escritura       1      352714  Wed May 23 13:38:40 2018
Ultima Escritura        50     26508   Wed May 23 13:38:42 2018
Menor Posición          26     9516    Wed May 23 13:38:41 2018
Mayor Posición          19     492352  Wed May 23 13:38:40 2018

PID: 16904
Numero de escrituras: 50
Primera Escritura       1      349011  Wed May 23 13:38:40 2018
Ultima Escritura        50     203121  Wed May 23 13:38:42 2018
Menor Posición          33     5039    Wed May 23 13:38:41 2018
Mayor Posición          24     491840  Wed May 23 13:38:41 2018

PID: 16905
Numero de escrituras: 50
Primera Escritura       1      493002  Wed May 23 13:38:40 2018
Ultima Escritura        50     323975  Wed May 23 13:38:42 2018
Menor Posición          39     18755   Wed May 23 13:38:42 2018
Mayor Posición          47     493487  Wed May 23 13:38:42 2018

PID: 16906
Numero de escrituras: 50
Primera Escritura       1      157979  Wed May 23 13:38:40 2018
Ultima Escritura        50     61521   Wed May 23 13:38:42 2018
Menor Posición          24     1348    Wed May 23 13:38:41 2018
Mayor Posición          41     496591  Wed May 23 13:38:42 2018

PID: 16907
Numero de escrituras: 50
Primera Escritura       1      120270  Wed May 23 13:38:40 2018
Ultima Escritura        50     69052   Wed May 23 13:38:42 2018
Menor Posición          48     2896    Wed May 23 13:38:42 2018
Mayor Posición          14     494242  Wed May 23 13:38:41 2018

PID: 16908
Numero de escrituras: 50
Primera Escritura       1      246291  Wed May 23 13:38:40 2018
Ultima Escritura        50     486024  Wed May 23 13:38:42 2018
Menor Posición          32     22037   Wed May 23 13:38:41 2018
Mayor Posición          38     488099  Wed May 23 13:38:42 2018
```

```
PID: 16910
Numero de escrituras: 50
Primera Escritura       1      198413  Wed May 23 13:38:40 2018
Ultima Escritura       50      177257  Wed May 23 13:38:43 2018
Menor Posición          7      3536    Wed May 23 13:38:40 2018
Mayor Posición         39      475887  Wed May 23 13:38:42 2018

PID: 16911
Numero de escrituras: 50
Primera Escritura       1      467343  Wed May 23 13:38:40 2018
Ultima Escritura       50      147583  Wed May 23 13:38:43 2018
Menor Posición         49      3892    Wed May 23 13:38:43 2018
Mayor Posición         12      490494  Wed May 23 13:38:41 2018

PID: 16912
Numero de escrituras: 50
Primera Escritura       1      293153  Wed May 23 13:38:40 2018
Ultima Escritura       50      344246  Wed May 23 13:38:43 2018
Menor Posición         40      6581    Wed May 23 13:38:42 2018
Mayor Posición          8      486867  Wed May 23 13:38:40 2018

PID: 16914
Numero de escrituras: 50
Primera Escritura       1      230073  Wed May 23 13:38:40 2018
Ultima Escritura       50      49452   Wed May 23 13:38:43 2018
Menor Posición          9      7582    Wed May 23 13:38:41 2018
Mayor Posición          8      499642  Wed May 23 13:38:41 2018

$ ./leer_sf disco

DATOS DEL SUPERBLOQUE
posPrimerBloqueMB = 1
posUltimoBloqueMB = 13
posPrimerBloqueAI = 14
posUltimoBloqueAI = 3138
posPrimerBloqueDatos = 3139
posUltimoBloqueDatos = 99999
posInodoRaiz = 0
posPrimerInodoLibre = 203
cantBloquesLibres = 88480 [12]
cantInodosLibres = 24797 [13]
totBloques = 100000
totInodos = 25000
```

[12] Dependerá en cada ejecución de las ubicaciones aleatorias de las escrituras

[13] Se han reservado 202 inodos: 1 para el directorio de simulación + 100 directorios de procesos + 100 ficheros prueba.dat + 1 fichero informe.txt

ANEXO 1

Makefile para compilar con la librería pthread:

```
CC=gcc
CFLAGS=-c -g -Wall -std=gnu99
LDFLAGS=-pthread

SOURCES=bloques.c ficheros_basico.c ficheros.c directorios.c mi_mkfs.c
leer_sf.c escribir.c leer.c truncar.c permitir.c mi_mkdir.c mi_chmod.c mi_ls.c
mi_link.c mi_escribir.c mi_cat.c mi_stat.c mi_rm.c semaforo_mutex_posix.c
simulacion.c verificacion.c
LIBRARIES=bloques.o ficheros_basico.o ficheros.o directorios.o
semaforo_mutex_posix.o
INCLUDES=bloques.h ficheros_basico.h ficheros.h directorios.h
semaforo_mutex_posix.h simulacion.h verificacion.h
PROGRAMS=mi_mkfs leer_sf escribir leer truncar permitir mi_mkdir mi_chmod
mi_ls mi_link mi_escribir mi_cat mi_stat mi_rm simulacion verificacion
OBJS=$(SOURCES:.c=.o)

all: $(OBJS) $(PROGRAMS)

$(PROGRAMS): $(LIBRARIES) $(INCLUDES)
	$(CC) $(LDFLAGS) $(LIBRARIES) $@.o -o $@

%.o: %.c $(INCLUDES)
	$(CC) $(CFLAGS) -o $@ -c $<

.PHONY: clean
clean:
	rm -rf *.o *~ $(PROGRAMS)
```

ANEXO 2

Pesquisas buscando mayor precisión en los sellos de las escrituras para una mejor discriminación entre la primera y la última en base al tiempo.

Obligar a que los sleeps se ejecuten realmente por el tiempo establecido.

Explorar la función gettimeofday() que permite una precisión de hasta microsegundos:

#include <sys/time.h>

int gettimeofday(struct timeval *_tv_, struct timezone *_tz_);

```
struct timeval {
   time_t      tv_sec;     /* seconds */
   suseconds_t tv_usec;    /* microseconds */
};
```

```
// Resta de dos fechas, en milisegundos, obtenidas con gettimeofday()
struct timeval tv;
gettimeofday(&tv, NULL);
// ...
struct timeval tv2;
gettimeofday(&tv2, NULL);
int microseconds = (tv2.tv_sec - tv.tv_sec) * 1000000 + ((int)tv2.tv_usec - (int)tv.tv_usec);
int milliseconds = microseconds/1000;
struct timeval tv3;
tv3.tv_sec = microseconds/1000000;
tv3.tv_usec = microseconds%1000000;
```

Ver también:
https://stackoverflow.com/questions/43732241/how-to-get-datetime-from-gettimeofday-in-c

```
time.tv_nsec=  200000000;

int sleep = nanosleep(&time, &rem);

while (sleep == -1){

     time.tv_nsec = rem.tv_nsec;

     sleep = nanosleep(&time, &rem);

}
```