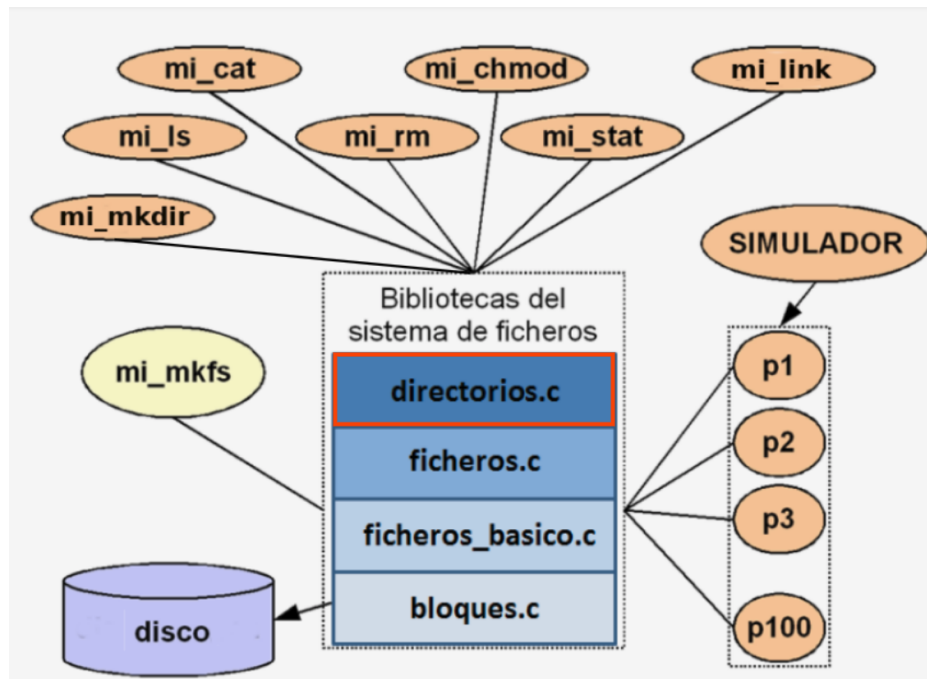


## Nivel 7: `directorios.c` {`extraer_camino()`, `buscar_entrada()`}

Ahora que ya tenemos finalizadas 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`)<sup>1</sup> estará formada por los siguientes campos:

Nombre de archivo/directorio	ninodo
------------------------------	--------

```
#define TAMNOMBRE 60 //tamaño del nombre de directorio o fichero, en ext2 = 256
struct entrada {
    char nombre[TAMNOMBRE];
    unsigned int ninodo;
};
```

<sup>1</sup> 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`.

Este campo *nombre* **no incluye la ruta** (ni el carácter de separación '/').  
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 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 es aquél que no tiene entrada de directorio.

Puede haber varios nombres de archivos, distribuidos 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 '/' (en tal caso **\*inicial** contendrá el nombre de un directorio).
- Cuando no hay segundo '/', copia **\*camino** en **\*inicial** sin el primer '/' (en tal caso **\*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.<sup>2</sup>
- 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` = "" //no puedo poner '\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 (identificado con `*p_inodo_dir`).

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:

---

<sup>2</sup> 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'.

- El número de inodo (`*p_inodo`) al que está asociado el nombre de la entrada buscada.
- El número de entrada (`*p_entrada`) dentro del inodo `*p_inodo_dir` que lo contiene (empezando por 0).

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 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)]`

En este ejemplo `BLOCKSIZE=256`, `sizeof(struct entrada)= 64`, así que tenemos 4 entradas de directorio por bloque.

Teniendo en cuenta que todos los bloques han de tener el mismo tamaño, los bloques de punteros deberían contener  $256/4 = 64$  elementos, pero por simplicidad del ejemplo, para hacer el seguimiento gráfico de cómo se buscan las entradas, se han considerado bloques de 4 punteros.

El struct inodo se compone, también 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`).

Número inodo	1	5	7	140	150	20	33
Punteros Directos	125	93	15	30	3	19	80
	120	20	18	37	32	22	75
Puntero Indirecto	150	10	NULL	28	NULL	33	45

También conocemos el contenido de los siguientes bloques de datos:

Numero Bloque	7	125	10	18	120	32	13
	14 pepe	1 .	27	50 Linux	7 bin	33 indice.txt	Futuras actualiza clones ...
	16 carta1	1 ..	34	60 tech-rep.	10/sbin	42 Backup	
	27 carta2	8 etc	40	65 juegos	15 software	31 Juan	
	32 carta4	120 home	41	150 publico	140 dev	17 Raul	

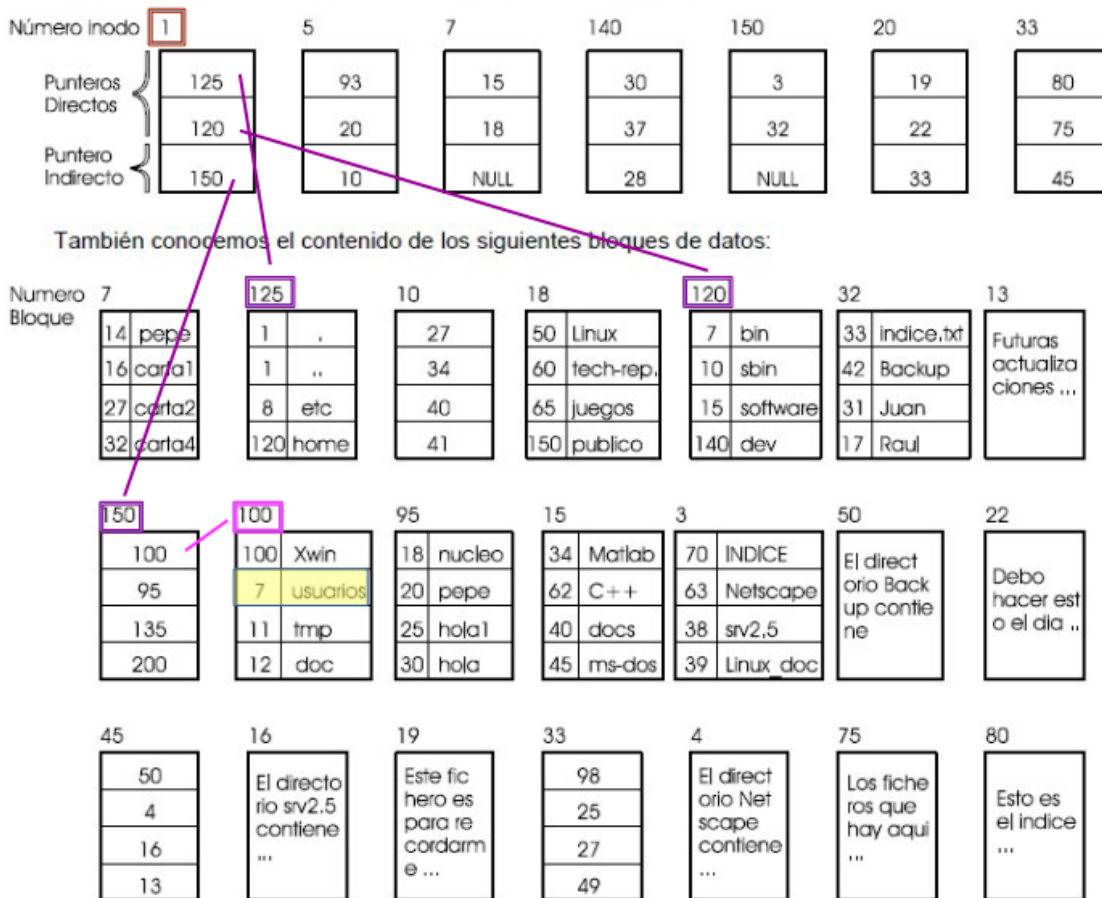
150	100	95	15	3	50	22
100	100 Xwin	18 nucleo	34 Matlab	70 INDICE	El direct	Debo
95	7 usuarios	20 pepe	62 C++	63 Netscape	orio Back	hacer est
135	11 tmp	25 hola1	40 docs	38 srv2.5	up conte	o el dia ..
200	12 doc	30 hola	45 ms-dos	39 Linux_doc	ne	

45	16	19	33	4	75	80
50	El directo	Este fic	98	El direct	Los fiche	Esto es
4	rio srv2.5	hero es	25	orio Net	ros que	el indice
16	contiene	para re	27	scape	hay aqui	...
13	...	cordarm	49	contiene	...	

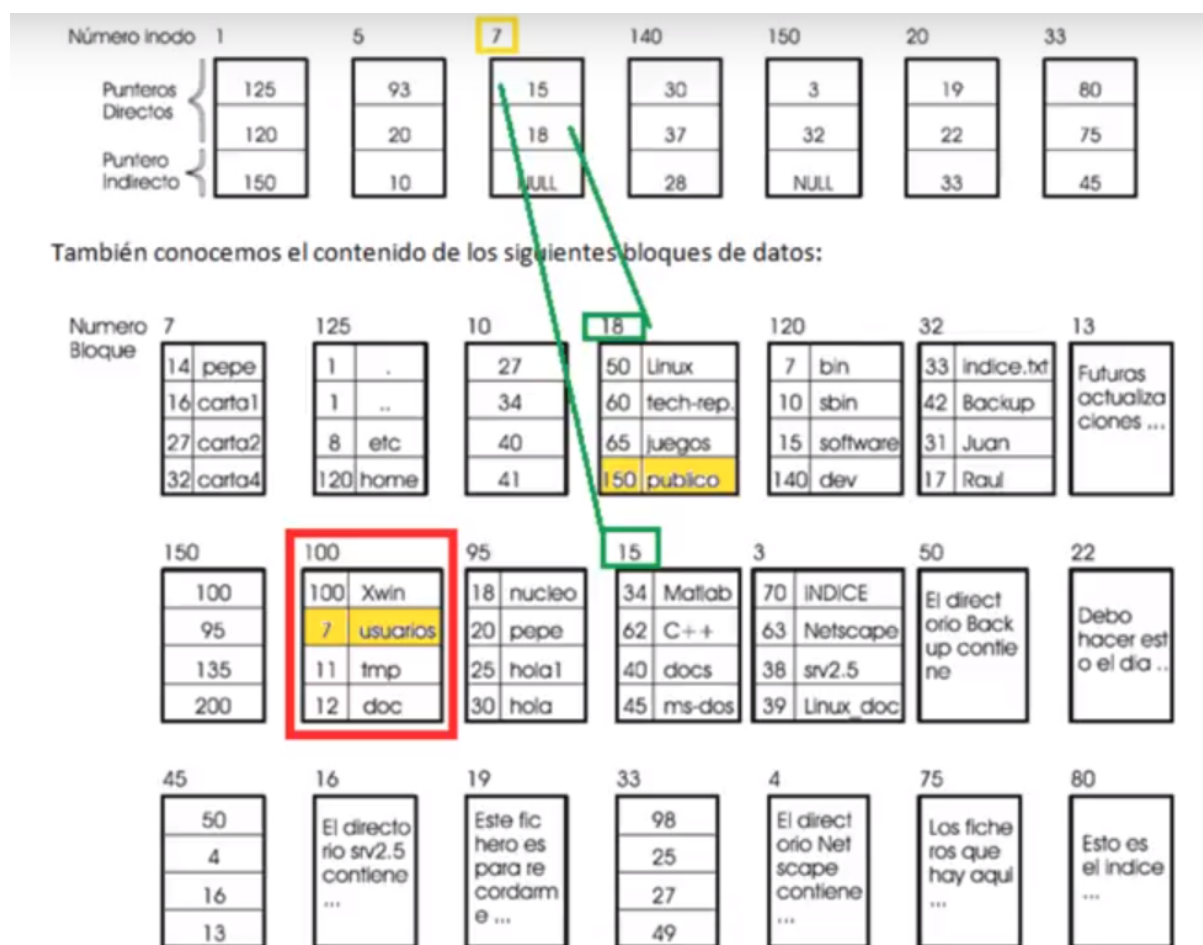
- 1ra llamada recursiva a buscar\_entrada():
  - Valores de entrada:
    - \*camino\_parcial = "/usuarios/publico/indice.txt"
    - \*p\_inodo\_dir = 1 (nº de inodo del directorio raíz "/" en este ejemplo)
  - Tras llamar a extraer\_camino():
    - \*inicial = "usuarios"
    - \*final = "/publico/indice.txt"
  - Objetivo:
    - obtener el nº de inodo (\*p\_inodo) asociado al nombre "usuarios" dentro del directorio "/": 7
    - y el nº de entrada correspondiente (\*p\_entrada)<sup>3</sup>: 9

<sup>3</sup> Empezamos a contar por la 0



- 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:
    - obtener el nº de inodo (\*p\_inodo) asociado al nombre "publico" dentro del directorio "usuarios": 150
    - y el nº de entrada correspondiente (\*p\_entrada)<sup>4</sup>: 7

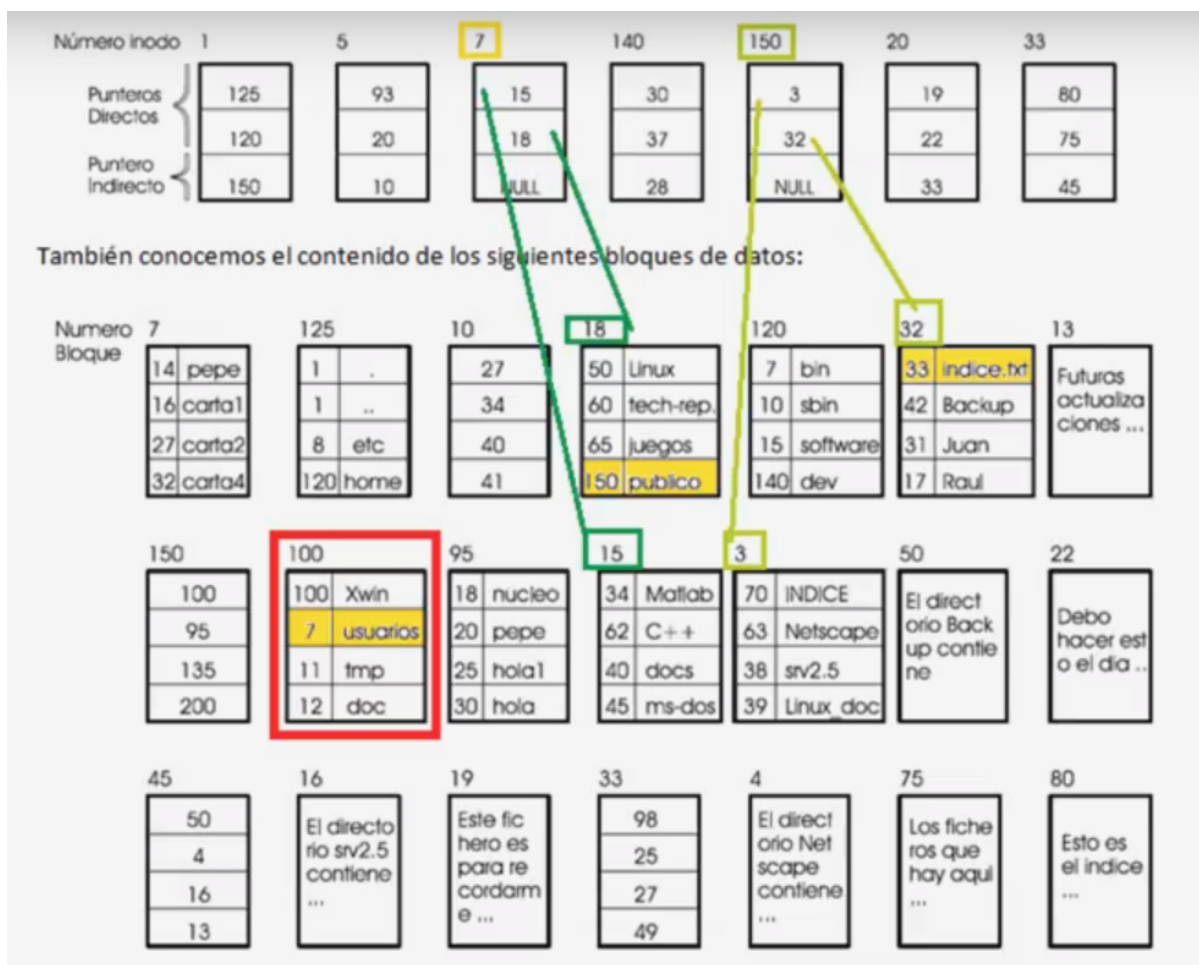
<sup>4</sup> Empezamos a contar por la 0



- 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
    - y el nº de entrada correspondiente (`*p_entrada`)<sup>5</sup>: 4

<sup>5</sup> Empezamos a contar por la 0





La misma función nos puede servir tanto para sólo **consultar** si **reservar=0** (llamadas desde `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()`<sup>6</sup>; 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()`.

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**:

<sup>6</sup> 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



- 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!!!**)<sup>7</sup>
- 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
var
    entrada: estructura entrada
    inodo_dir: estructura inodo
    inicial[sizeof(entrada.nombre)]: car
    final[strlen(camino_parcial)]: car
    tipo: car
    cant_entradas_inodo, num_entrada_inodo: int
fvar

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

extraer_camino (camino_parcial, inicial, final, &tipo)
si (error al extraer camino) entonces devolver ERROR_CAMINO_INCORRECTO fsi

//buscamos la entrada cuyo nombre se encuentra en inicial
leer_inodo( *p_inodo_dir, &inodo_dir)
si (inodo_dir no tiene permisos de lectura) entonces
    devolver ERROR_PERMISO_LECTURA
fsi

inicializar el buffer de lectura con 0s
//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
```

<sup>7</sup> 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

```
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
    mientras ((num_entrada_inodo < cant_entradas_inodo) y (inicial ≠ entrada.nombre)) hacer
        num_entrada_inodo++
        leer siguiente entrada //previamente volver a inicializar el buffer de lectura con 0s
    fmientras
fsi

si (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
                    reservar un inodo como directorio y asignarlo a la entrada
                    si_no //cuelgan más directorios 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 en el directorio padre
            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
    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 o fichero creado o leído
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 encontrada;
    devolver buscar_entrada (final, p_inodo_dir, p_inodo, p_entrada, reservar, permisos)
fsi
devolver EXIT_SUCCESS
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
```

Ejemplo 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 adaptar `leer_sf.c` añadiendo el código siguiente para hacer algunas pruebas de creación de ficheros y directorios con `buscar_entrada()` y detectar algunos errores<sup>8</sup>:

```
#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){
    ...
    //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
    ...
}
```

Resultado de la ejecución<sup>9</sup>:

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

<sup>8</sup> Los errores relacionados con permisos los testaremos en el siguiente nivel

<sup>9</sup> Aquí ya se pueden ocultar los `fprintf()` de `traducir_bloque_inodo()`

```
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
```

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.

\*\*\*\*\*

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]
*****
```

