Sistemes Operatius II

Desenvolupament d'un Sistema de Fitxers

Andreu Cortès Vicens
43.233.137 Z
GEIN

Funcionament

El funcionament de l'aplicació és relativament senzill respecte la seva implementació.
El que ofereix aquesta aplicació són múltiples operacions sobre un Dispositiu Virtual.

Operacions Possibles (al més alt nivell)

Creació (mi_mkfs.c)

```
./mi_mkfs [nom_del_dispositiu] [quantitat_de_blocs]
```

En primer lloc s'especifiquen un nom per al Dispositiu i el nombre de blocs de 1024 Bytes que es volen. D'aquesta manera es crearà un dispositiu de tants blocs com es vulgui. Per tant, tindrem un arxiu en el nostre Directori de Context el qual estarà compost per tants de Bytes com el producte entre el nombre facilitat i el nombre de Bytes per Bloc.

Al més baix nivell, el que fa aquest programa és una iteració inicialitzant un buffer en blanc, i escrivint-lo en el dispositiu (a posicions de memòria consecutives), un darrere l'altre. Finalitza l'execució quan s'ha finalitzat la iteració.

Creació de Directoris (mi_mkdir.c)

```
./mi_mkdir [nom_del_dispositiu] [permisos] [cami]
```

S'especificarà el nom del dispositiu a modificar i la ruta d'un directori o fitxer nou, a més dels permisos que es vulguin donar a la creació.
En un mètode recursiu (buscar_entrada), es cercarà el camí que duu al nou fitxer o directori i si tot és correcte (si els directoris intermedis tenen permisos i existeixen, el fitxer o directori no existeixi, i queda lloc per escriure) es crearà. Ara bé, si hi ha hagut alguns dels errors anteriors, la creació es veurà avortada, i no s'haurà creat.

Llistat de Contingut de Directoris (mi_ls.c)

```
./mi_ls [nom_del_dispositiu] [cami]
```

S'especificarà el nom del dispositiu a modificar i la ruta d'un directori.
Primer es comprovarà que el directori existeix, i llavors en un recorregut per tot el contingut del directori s'aniran llistant les entrades en un buffer. Finalment el que es farà és un recorregut per tot el buffer i s'anirà imprimint la informació per pantalla.

Obtenir Informació d'un Directori o Fitxer (mi_stat.c)

```
./mi_stat [nom_del_dispositiu] [cami]
```

S'especificarà el nom del dispositiu a modificar i la ruta d'un fitxer o directori.
Es comprovarà que existeix el camí facilitat (amb buscar_entrada) i llavors es llistarà la informació més rellevant del Inodo.

Canviar els permisos d'un Directori o Fitxer (mi_chmod.c)

```
./mi_chmod [nom_del_dispositiu] [permisos] [cami]
```

S'especificarà el nom del dispositiu a modificar i la ruta del fitxer o directori a canviar els permisos. Es comprovarà que existeix un fitxer o directori que correspongui al camí, i llavors amb la funció mi_chmod_f es canviarà el camp corresponent de l'Inodo al que apunta.

Llegir tot el Contingut d'un Fitxer (mi_cat.c)

```
./mi_cat [nom_del_dispositiu] [cami]
```

S'especificarà el nom del dispositiu on es troba el fitxer a llegir.
Es comprovarà que existeix i seguidament es farà una iteració per tota la llargària del fitxer,traduint els blocs lògics en blocs físics (amb traducir_bloque_inodo i traducir_recursivo) i imprimint el contingut per pantalla.

Llegir la informació del SuperBloc, Mapa de Bits i Array d'Inodos (leer_SF.c)

```
./leer_SF [nom_del_dispositiu]
```

S'especificarà el nom del dispositiu a revisar. Es llegirà el SuperBloc, i se'n llistarà la informació. Llavors es llegiran tots els Inodos ocupats i se'n mostraran les informacions més rellevants com: el Nombre, el Tamany en Bytes Lògics i les dates adients (atime, mtime, ctime).

Vincular dos fitxers (mi_ln.c)

```
./mi_ln [nom_del_dispositiu] [enllaç_origen] [enllaç_destí]
```

S'especificarà el nom del dispositiu, el nom del ftxer origen i el nom del fitxer destí, el qual s'ha de copiar. Primer de tot es cercarà el fitxer origen (amb buscar_entrada) i es comprovarà que existeix. Llavors, es veurà si el fitxer destí existeix o no, i si no es troba, se'n crearà un de nou. Si es trobava, s'alliberarà l'entrada del directori pare i es canviarà per la corresponent. Es limitarà aquesta funció a fitxers, i així no permetrem que hi hagi cicles en l'arbre de directoris.

Esborrar una entrada (mi_rm.c)

```
./mi_rm [nom_del_dispositiu] [enllaç]
```

S'especificarà el nom del dispositiu i el nom de l'entrada a esborrar.
Es comprovarà que sigui un fitxer o un directori buit. Si no és cap d'aquests dos casos, no podrem seguir. Primer, es cercarà l'entrada al dispositiu (mitjançant buscar_entrada). Llavors es veurà si es pot esborrar, i finalment traurem l'entrada corresponent del directori pare, a més d'alliberar l'Inodo.

Simulació (simulacion.c)

```
./simulacion [nom_del_dispositiu]
```

S'especificarà el nom del dispositiu i seguidament es procedirà a l'escriptura dels diferents directoris i fitxers de manera recurrent, tenint en compte la concurrència entre els processos que s'aniran executant a la vegada. Es crearà un directori anomenat “simul_aaaammddhhmmss/” i dins aquest hi aniran 100 directoris de la forma “/proces_xxxx/” on 'xxxx' és el nombre PID del Procés. Dins cada un d'aquests directoris hi anirà un fitxer anomenat “prueba.dat” que contindrà 50 registres de l'escriptura, que emmagatzema informació sobre cada escriptura.

Verificació (verificacion.c)

```
./verificacion[nom_del_dispositiu]
```

S'especificarà el nom del dispositiu a verificar les escriptures. Es llegeix el primer directori, i es van llegint cada entrada fins a trobar cada fitxer. Llavors, es fa una iteració per tots els blocs que estan ocupats de cada fitxer, i finalment, es mostra cada registre per pantalla.

Algoritmes Interessants

verificació:

És un algoritme que cerca els 50 registres escrits per l'algoritme de Simulació. Llegeix cada fitxer, i després en fa un recorregut cercant els registres amb dades més interessants com: el que s'ha escrit en la posició més baixa i més alta, i el que ha estat el primer i el darrer en escriure. Com que l'algoritme de Simulació és relativament ràpid, es podrà comprovar que gairebé no hi ha diferència en els temps.

buscar_entrada:

És un algoritme que funciona amb diferents mètodes, i el que fa és un recorregut per tot l'Inodo de tipus directori, fins a trobar l'entrada corresponent que ha estat passada per paràmetre. Com es podrà comprovar, aquesta solució d'aquest algoritme és un poc més eficient que l'algoritme donat, ja que es llegeix l'Inodo per Blocs, i no es va a cercar cada entrada d'una en una. Així, en un accés al disc accedim a 16 entrades, mentre que de l'altre manera, accediríem 16 vegades al disc en un espai de memòria consecutiu.

liberar_bloques_inodo:

Aquest és un algoritme que, en la meva solució, ha optat per ser recursiu (es necessita d'un mètode afegit). El que es fa és, recursivament, a través dels nivells de Punters Indirectes de l'Inodo, anar a cercar cada bloc, i alliberant-lo i esborrant la seva informació.

traducir_bloque_inodo:

Aquest és un algoritme, també solucionat de manera recursiva, que va cercant blocs en els diferents nivells de Punters Indirectes. Segons amb els paràmetres amb que ha estat cridat, pot reservar un bloc nou. Noti's l'agilització de la tria de casos: una Matriu on segons els valors de les entrades, podem saber directament (amb cost O(1)) el camí que hem de recórrer. En canvi, si s'optàs per una estructura de comprovacions amb clàusules 'if', es necessitarien quatre o més sentències per aconseguir la mateixa finalitat.

Mostra de Funcionament

Seguidament es mostren una sèrie de captures de pantalla on es demostra la sortida de cada un dels programes:

mi_mkfs:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_mkfs disco.imagen 100000
El dispositiu s'ha muntat correctament.
cva@cva-X71Sr:~/Documentos/Practica$
```

mi_mkdir:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_mkdir disco.imagen 7 /dir1/
cva@cva-X71Sr:~/Documentos/Practica$
```

mi_ls:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_ls disco.imagen /
rwx     dir1/           0              2013-08-22 18:49:13
cva@cva-X71Sr:~/Documentos/Practica$
```

mi_chmod:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_chmod disco.imagen 3 /dir1/
cva@cva-X71Sr:~/Documentos/Practica$
```

mi_stat:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_stat disco.imagen /dir1/
        Tipus:                                  Directori
        Permisos:                               -wx
        Nombre d'Entrades:                      1
        Blocs Ocupats:                          0
        Tamany en Bytes Lògics:                 0
        Darrer Accés a les Dades:               2013-08-22 18:49:13
        Darrera Modificació de les Dades:       2013-08-22 18:49:13
        Darrera Modificació de l'Inodo:         2013-08-22 18:53:15

cva@cva-X71Sr:~/Documentos/Practica$
```

mi_mkdir + mi_escribir:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_mkdir disco.imagen 7 /fich1
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_escribir disco.imagen 60000 /fich1 "Hola. Buenas Tardes."
cva@cva-X71Sr:~/Documentos/Practica$
```

mi_cat:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_cat disco.imagen /fich1
Hola. Buenas Tardes.
cva@cva-X71Sr:~/Documentos/Practica$
```

leer_SF:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./leer_SF disco.imagen
        INFORMACIÓ del SUPERBLOQUE.
        Blocs del Mapa de Bits: Primer: 1; Darrer: 13.
        Blocs de l'Array d'Inodos: Primer: 14; Darrer: 3138.
        Blocs del Bloc de Dades: Primer: 3139; Darrer: 99999.
        Blocs de Dades Lliures: 96858. Totals: 100000.
        Nombre d'Inodos Lliures: 24997. Totals: 25000.
        Inodo Directori Arrel: 0.

        sizeof(struct Inodo) = 128.

        INFORMACIÓ del MAPA DE BITS.
        Hi ha Ocupats 96858 de 100000 Blocs.

        INFORMACIÓ dels INODOS Ocupats.
ID: 0           Tamany: 128             ATIME: Thu 2013-08-22 18:58:15   MTIME: Thu 2013-08-22 18:55:52   CTIME: Thu 2013-08-22 18:55:52
ID: 1           Tamany: 0               ATIME: Thu 2013-08-22 18:49:13   MTIME: Thu 2013-08-22 18:49:13   CTIME: Thu 2013-08-22 18:53:15
ID: 2           Tamany: 60020           ATIME: Thu 2013-08-22 18:58:15   MTIME: Thu 2013-08-22 18:56:25   CTIME: Thu 2013-08-22 18:56:25

cva@cva-X71Sr:~/Documentos/Practica$
```

mi_ln:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_ln disco.imagen /fich1 /fich2
cva@cva-X71Sr:~/Documentos/Practica$
```

mi_rm:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./mi_rm disco.imagen /fich1
cva@cva-X71Sr:~/Documentos/Practica$
```

simulacion:

```
cva@cva-X71Sr:~/Documentos/Practica$ ./simulacion disco.imagen
Processos Acabats: 1 de 100.
Processos Acabats: 2 de 100.
Processos Acabats: 3 de 100.
Processos Acabats: 4 de 100.
Processos Acabats: 5 de 100.
Processos Acabats: 6 de 100.
Processos Acabats: 7 de 100.
Processos Acabats: 8 de 100.
```

```
Processos Acabats: 96 de 100.
Processos Acabats: 97 de 100.
Processos Acabats: 98 de 100.
Processos Acabats: 99 de 100.
Processos Acabats: 100 de 100.
Final de la Simulació.
cva@cva-X71Sr:~/Documentos/Practica$
```

verificacion:

```
Verificació del procés 6038:
        Nombre de Validacions: 50.
        Informació dels Registres:
                Posició menor = 54609.
                Posició major = 494578.
                Temps primer = 2013-08-22 19:02:29
                Temps darrer = 2013-08-22 19:04:01
Verificació del procés 6040:
        Nombre de Validacions: 50.
        Informació dels Registres:
                Posició menor = 3368.
                Posició major = 491341.
                Temps primer = 2013-08-22 19:02:29
                Temps darrer = 2013-08-22 19:04:01
cva@cva-X71Sr:~/Documentos/Practica$
```

Annex

Codi de la Pràctica

Arxiu “bloques.c”

```
#include "bloques.h"

static int descriptor = 0;
static sem_t *mutex;

int bmount(const char *camino){
      /* Funció que munta un dispositiu: si està creat, l'obre;
       * si no, el crea de bell nou. */
      descriptor =  open(camino, O_RDWR|O_CREAT, 0666);
      if(descriptor == -1) printf("bloques.c, 10: Error en obrir o
            crear el dispositiu: %s.\n", camino);
      mutex = initSem();
      return descriptor;
}

int bumount(){
      /* Funció que desmunta un dispositiu ja creat i obert. */
      int d;
      d = close(descriptor);
      if(d == -1) printf("bloques.c, 19: Error en tancar el dispositiu.\n");
      deleteSem();
      return d;
}

int bwrite(unsigned int bloque, const void *buf){
      /* Funció que escriu el contingut d'un buffer de memòria en un
       * bloc de dades que ha estat indicat. */
      int d;
      lseek(descriptor, bloque*BLOCKSIZE, SEEK_SET);
      d = write(descriptor, buf, BLOCKSIZE);
      if(d == -1) printf("bloques.c, 30: Error a l'escriure
            en el bloc: %i.\n", bloque);
      return d;
}

int bread(unsigned int bloque, void *buf){
      /* Funció que llegeix el contingut d'un bloc de dades del
       * dispositiu i el retorna pel buffer passat per paràmetre. */
      int d;
      lseek(descriptor, bloque*BLOCKSIZE, SEEK_SET);
      d = read(descriptor, buf, BLOCKSIZE);
      if(d == -1) printf("bloques.c, 40: Error en llegir
            el bloc: %i.\n", bloque);
      return d;
}

/* Funcions pròpies per al control dels Semàfors. */
void mi_waitSem(){
      waitSem(mutex);
}

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

Arxiu “bloques.h”

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <math.h>
#include <time.h>
#include "semaforo_mutex_posix.h"

#define BLOCKSIZE 1024

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

Arxiu “ficheros_basico.c”

```
#include "ficheros_basico.h"

int tamMB(unsigned int nbloques){
	/* Funció que retorna el nombre de Blocs que ha de tenir
	 * el Mapa de Bits del dispositiu.*/
	int n = nbloques / 8;
	if(n % BLOCKSIZE == 0) return (n / BLOCKSIZE);
	else return ((n / BLOCKSIZE) + 1);
}

int tamAI(unsigned int ninodos){
	/* Funció que retorna el nombre de Bloc que ha de tenir
	 * l'Array d'Inodos del dispositiu. */
	int n = sizeof(struct Inodo) * ninodos;
	if(n % BLOCKSIZE == 0)return (n / BLOCKSIZE);
	else return ((n / BLOCKSIZE) + 1);
}

int initSB(unsigned int nbloques, unsigned int ninodos){
	/* Funció que inicialitza el SuperBloque del dispositiu
	 * segons el nombre de blocs totals (nbloques) i
	 * el nombre d'inodos (ninodos). */

	struct SB a;

	/* Nombre dels Blocs Delimitadors de les parts. */
	a.PrimerBloqueMB = 1;
	a.UltimoBloqueMB = 1 + tamMB(nbloques) - 1;
	a.PrimerBloqueAI = a.UltimoBloqueMB + 1;
	a.UltimoBloqueAI = 1 + tamMB(nbloques) + tamAI(ninodos) - 1;
	a.PrimerBloqueData = a.UltimoBloqueAI + 1;
	a.UltimoBloqueData = nbloques - 1;

	a.InodoDirectorioRaiz = 0;
	a.PrimerInodoLibre = 0;

	a.BloquesLibres = nbloques;
	a.InodosLibres = ninodos;

	a.TotalBloques= nbloques;
	a.TotalInodos = ninodos;

	memset(&a.padding, 0, sizeof(a.padding));

	if(bwrite(SBpos, &a) == -1){
		printf("ficheros_basico.c, 46: Error a l'escriure al
			dipositiu a la posició: %i.\n", SBpos);
		return -1;
	}
	return 0;
}
```

```
int initMB(unsigned int nbloques){
	/* Funció que inicialitza el Mapa de Bits del dispositiu. */

	struct SB a;
	if(bread(SBpos, &a) == -1){
		printf("ficheros_basico.c, 57: Error al llegir el SB.\n");
		return -1;
	}

	/* Definim un buffer i l'inicialitzam a 0. */
	unsigned char buffer[BLOCKSIZE];
	memset(buffer, 0, BLOCKSIZE);

	/* Fem un recorregut per tots els blocs del MB col·locant el
	 * buffer ple de zeros al MB. */
	int i = 0;
	for(i = a.PrimerBloqueMB; i < a.PrimerBloqueAI; i++){
		if(bwrite(i, buffer) == -1){
			printf("ficheros_basico.c, 70: Error a l'escriure al
				dispositiu a la posició: %i.\n", i);
			return -1;
		}
	}

	/* Posam els valors al Mapa de Bits que corresponen als Blocs
	 * ocupats pel SuperBloque, Mapa de Bits i Array de Inodos. */
	for(i = 0; i < a.PrimerBloqueData; i++){
		if(escribir_bit(i, 1) == -1){
			printf("ficheros_basico.c, 79: Error en escriure a 1 el bit:
				%i.\n", i);
			return -1;
		}
	}
	return 0;
}
```

```
int initAI(){
	/* Funció que inicialitza l'Array d'Inodos. */

	/* Llegim el SuperBloque. */
	struct SB a;
	if(bread(SBpos,&a) == -1){
		printf("ficheros_basico.c, 92: Error al llegir el SB.\n");
		return -1;
	}

	/* Iteram per tots els Inodos del dispositiu:
	 *		Per cada Bloc de l'Array d'Inodos entram dins cada un
	 *		dels 8 Inodos de cada Bloc.
	 *		I els inicialitzam.
	 * Cada Inodo Lliure contindrà en el punteroDirecto[0] un punter
	 * al següent Inodo Lliure. */
	int i = 0, j = 0, x = 1;
	struct Inodo inodos[BLOCKSIZE/(sizeof(struct Inodo))];
	memset(inodos, 0, BLOCKSIZE);
	for(i = a.PrimerBloqueAI; i <= a.UltimoBloqueAI; i++){
		for(j = 0; j < BLOCKSIZE/(sizeof(struct Inodo)); j++){
			inodos[j].tipo = 'l';
			if(x < a.TotalInodos){
				inodos[j].punterosDirectos[0] = x;
				x++;
			}
			else{
				inodos[j].punterosDirectos[0] = UINT_MAX;
				break;
			}
		}
		if(bwrite(i,&inodos) == -1){
			printf("ficheros_basico.c, 118: Error a l'escriure al
				dispositiu a la posició: %i.\n", i);
			return -1;
		}
		memset(inodos, 0, BLOCKSIZE);
	}
	if(bwrite(SBpos, &a) == -1) {
		printf("ficheros_basico.c, 125: Error a l'escriure el SB.\n");
		return -1;
	}
	return 0;
}
```

```
int escribir_bit(unsigned int nbloque, unsigned int bit){
      /* Funció en la que escriurem el bit del bloc (nbloque) que
       * vulguem amb el signe (bit) que vulguem. */
      int e = 0;

      /* Llegim el SuperBloque. */
      struct SB a;
      if(bread(SBpos, &a) == -1){
            printf("ficheros_basico.c, 139: Error en llegir SB.\n");
            return -1;
      }

      int posbyte = nbloque / 8;
      int posbit = nbloque % 8;
      int numbloque = posbyte / BLOCKSIZE + a.PrimerBloqueMB;
      posbyte = posbyte % BLOCKSIZE;

      unsigned char buf[BLOCKSIZE];

      unsigned char mascara = 128;
      mascara >>= posbit;

      if(numbloque == 0){
            printf("ficheros_basico.c, 160: Error en bloc SB.\n");
            return -1;
      }

      /* Llegirem el Bloc en el que hem de canviar el bit de valor. */
      if(bread(numbloque, buf) == -1){
            printf("ficheros_basico.c, 166: Error en llegir el
                  bloc: %i.\n", numbloque);
            return -1;
      }

      switch(bit){
            case 0: buf[posbyte]  &= ~mascara;
                        a.BloquesLibres++;
                        break;
            case 1: buf[posbyte] |= mascara;
                        a.BloquesLibres--;
                        break;
            default:
                        printf("ficheros_basico.c, 181: El bit no té
                              el valor apropiat.\n");
                        return -1;
      }

      /* Primer escriurem el Bloc del Mapa de Bits ja canviat i
       * després el SuperBloc de manera que tot quedi actualitzat. */
      if(bwrite(numbloque, buf) == -1){
            printf("ficheros_basico.c, 188: Error en escriure
                  el bloc: %i.\n", numbloque);
            return -1;
      }

      if(bwrite(SBpos, &a) == -1){
            printf("ficheros_basico.c, 193: Error en escriure
                  el bloc del SB.\n");
            return -1;
      }
      return 0;
}
```

```
unsigned char leer_bit(unsigned int nbloque){
      /* Funció que retorna el valor del Bloc del Mapa de Bits
       * corresponent al número de Bloc (nbloque).
       * El valor retornat és un unsigned char, però és fàcilment
       * intercanviable amb un càsting (int). */
      struct SB a;
      if(bread(SBpos, &a) == -1){
            printf("ficheros_basico.c, 206: Error en llegir SB.\n");
            return -1;
      }
      /* Calculam la posició del bit que haurem de llegir.
      * A posbyte hi ha la posició del byte, amb la que calcularem
      * el Bloc que hem de llegir.
      * A posbit hi ha la posició del bit dins el byte que haurem
      * de canviar. */
      int posbyte = nbloque / 8;
      int posbit =  nbloque % 8;
      int numbloque = posbyte/BLOCKSIZE + a.PrimerBloqueMB;
      posbyte = posbyte % BLOCKSIZE;

      unsigned char buf[BLOCKSIZE];

      unsigned char mascara = 128;
      mascara >>= posbit;

      /* Llegim el bloc que necessitam i llavors calculam el valor
       * que quedarà emmagatzemat a màscara. */
      if(bread(numbloque, buf) == -1){
            printf("ficheros_basico.c, 228: Error en llegir
                  el bloc: %i.\n", numbloque);
            return -1;
      }

      mascara &= buf[posbyte];
      mascara >>= (7- posbit);

      return mascara;
}
```

```
int reservar_bloque(){
	/* Funció que reservarà un Bloc de Dades i retornarà
	 * el número del Bloc reservat. */

	struct SB a;
	if(bread(SBpos, &a) == -1){
		printf("ficheros_basico.c, 243: Error en llegir el SB. \n");
		return -1;
	}

	if(a.BloquesLibres <= 0){
		printf("ficheros_basico.c, 250: No hi ha més blocs disponibles.\n");
		return -1;
	}

	int i = a.PrimerBloqueMB - 1;
	unsigned char b[BLOCKSIZE];
	memset(b, 255, BLOCKSIZE);
	unsigned char bAux[BLOCKSIZE];
	memset(bAux, 255, BLOCKSIZE);

	/* Llegim el primer bloc del Mapa de Bits. Llavors,
	 * iterativament anirem comprovant si hi ha algun zero. */
	while(memcmp(b, bAux, BLOCKSIZE) == 0){
		i++;
		if(bread(i, b) == -1){
			printf("ficheros_basico.c, 265: Error en
				llegir el bloc: %i. \n", i);
			return -1;
		}
	}
	/* Ara, la variable 'i' conté el valor del primer bloc del
	 * Mapa de Bits que conté, almenys, un 0.
	 * I 'b' conté el bloc sencer. */

	unsigned char mascara = 128;
	int posbyte = 0, posbit = 0;
	while(b[posbyte] == 255){
		posbyte++;
	}
	/* Ara ja tenim la posició del byte on es troba el 0. */

	while(b[posbyte] & mascara){
		posbit++;
		mascara >>= 1;
	}
	/* Finalitzat aquesta iteració, a posbit tenim el nombre de
	 * bits del byte que no són 0 davant del primer. */

	int numbloque = ((i - a.PrimerBloqueMB)*8)*1024 + posbyte * 8 + posbit;
	if(escribir_bit(numbloque, 1) == -1){
		printf("ficheros_basico.c, 289: Error en escriure
			el bit: %i.\n", numbloque);
		return -1;
	}
	/* El control de l'increment i el decrement dels blocs ocupats
	 * es duu a terme en un nivell més baix: escribir_bit.
	 * Ja que es controla amb el valor del bit el nombre de Blocs
	 * ocupats i lliures. */

	return numbloque;
}
```

```
int liberar_bloque(unsigned int nbloque){
      /* Funció que allibera un Bloc de Dades ocupat o no, donant el
       * valor 0 en el Mapa de Bits corresponent al bloc. */
      if(escribir_bit(nbloque, 0) == -1){
            printf("ficheros_basico.c, 304: Error en escriure
                  el bit: %i.\n", nbloque);
            return -1;
      }
      return nbloque;
}

int escribir_inodo(struct Inodo in, unsigned int ninodo){
      /* Funció que escriu un Inodo en la posició que és especificada
       * per ninodo. */

      /* Llegim el SuperBloque. */
      struct SB a;
      if(bread(SBpos, &a) == -1){
            printf("ficheros_basico.c, 317: Error en llegir el SB.\n");
            return -1;
      }

      /* Calculam la Posició de l'Inodo: primer el Bloc on es troba i
       * i llavors la posició dins el Bloc. */
      int posBloque = ninodo / (BLOCKSIZE / (sizeof(struct Inodo))) +
                  a.PrimerBloqueAI;
      int posInodo = ninodo % (BLOCKSIZE / (sizeof(struct Inodo)));

      if(posBloque == 0){
            printf("ficheros_basico.c, 327: Error en Bloc del SB.\n");
            return -1;
      }

      struct Inodo inodo[BLOCKSIZE / (sizeof(struct Inodo))];
      if(bread(posBloque, inodo) == -1){
            printf("ficheros_basico.c, 333: Error en llegir
                  el Bloc d'Inodos.\n");
            return -1;
      }

      /* Assignam a l'Inodo del Bloc, l'Inodo que hem passat per
       * paràmetre, i escrivim aquest en el bloc corresponent. */
      inodo[posInodo] = in;

      if(bwrite(posBloque, inodo) == -1){
            printf("ficheros_basico.c, 342: Error en escriure
                  al Bloc d'Inodos.\n");
            return -1;
      }
      return ninodo;
}
```

```
struct Inodo leer_inodo(unsigned int ninodo){
      /* Funció que retorna l'Inodo apuntat per ninodo. */

      struct Inodo in;
      struct SB a;
      if(bread(SBpos, &a) == -1){
            printf("ficheros_basico.c, 354: Error en llegir el SB.\n");
            return in;
      }

      /* Hem de comprovar que l'Inodo al que apunta és possible de
       * trobar. Sinó, botarà un error. */
      if(ninodo >= a.TotalInodos){
            printf("ficheros_basico.c, 361: No existeix aquest
                  Inodo: %i.\n", ninodo);
            return in;
      }

      /* Calculam la posició en la que es troba i llegim el Bloc
       * corresponent. Finalment, retornam l'Inodo calculat. */
      int posBloque = ninodo / (BLOCKSIZE / (sizeof(struct Inodo))) +
                        a.PrimerBloqueAI;
      int posInodo = ninodo % (BLOCKSIZE / (sizeof(struct Inodo)));

      struct Inodo inodos[BLOCKSIZE / (sizeof(struct Inodo))];
      if(bread(posBloque, inodos) == -1){
            printf("ficheros_basico.c, 372: Error en llegir
                  el Bloc d'Inodos.\n");
            return in;
      }

      return inodos[posInodo];
}
```

```
int reservar_inodo(unsigned char tipo, unsigned char permisos){
	/* Funció que reserva el primer Inodo Lliure i
	 * retorna el punter a aquest. */
	int i = 0, j = 0;

	/* Llegim el SuperBloque per tenir la darrera informació
	 * del sistema. */
	struct SB a;
	if(bread(SBpos, &a) == -1){
		printf("ficheros_basico.c, 388: Error en llegir el SB.\n");
		return -1;
	}

	/* Hem de comprovar que hi hagi Inodos Lliures, perquè si no
	 * no se'n podria reservar cap més. */
	if(a.InodosLibres < 1){
		printf("ficheros_basico.c, 395: No hi ha cap Inodo Lliure.\n");
		return -1;
	}

	/* Llegim el primer Inodo Lliure que marca el SuperBloque. */
	struct Inodo in;
	in = leer_inodo(a.PrimerInodoLibre);

	/* Inicialitzam el contingut de l'Inodo, i guardam el punter al
	 * següent Inodo Lliure dins la variable 'j'. */
	in.tamBytesLogicos = 0;
	in.EntradasDirectorio = 1;

	j = a.PrimerInodoLibre;
	a.PrimerInodoLibre = in.punterosDirectos[0];

	for(i = 0; i < 12; i++){
		in.punterosDirectos[i] = 0;
	}
	for(i = 0; i < 3; i++){
		in.punterosIndirectos[i] = 0;
	}
	in.atime = time(NULL);
	in.mtime = time(NULL);
	in.ctime = time(NULL);
	in.tipo = tipo;
	in.permisos = permisos;

	/* Escrivim l'Inodo en la posició que toca.
	 * Després actualitzam la informació del SuperBloque. */
	if(escribir_inodo(in, j) == -1) {
		printf("ficheros_basico.c, 426: Error en escriure l'Inodo.\n");
		return -1;
	}

	a.InodosLibres--;
	if(bwrite(SBpos, &a) == -1){
		printf("ficheros_basico.c, 432: Error en escriure el SB.\n");
		return -1;
	}
	return j;
}
```

```
int traducir_recursivo(struct Inodo *in, int nivel, int pBSig, int blogico,
      unsigned int *bfisico, char reservar){

      /* Funció auxiliar recursiva per a la funció de
       * traducir_bloque_inodo mitjançant una recursió a través dels
       * nivells de profunditat que necessitam. */

      int i = 0, blogicoAux;
      unsigned int buf[BLOCKSIZE/(sizeof(unsigned int))];

      if(bread(pBSig, buf) == -1){
            printf("ficheros_basico.c, 446: Hi ha hagut error en llegir
                  el bloc: %i.\n", pBSig);
            return -1;
      }
      int j = (int)pow((BLOCKSIZE/sizeof(unsigned int)), nivel);
      blogicoAux = blogico / j;

      int r = 0, e = 0;
      if(reservar == 1) r = 1;
      if(buf[blogicoAux] != 0) e = 1;
      int matOpciones[2][2] = {-1, 0, 1, 0};

      switch(matOpciones[r][e]){
            case -1:    nivel = 0; i = 0;
                        break;
            case 1:     buf[blogicoAux] = reservar_bloque();
                        in->BloquesDataOcupados++;
                        if(bwrite(pBSig, buf) == -1){
                              printf("ficheros_basico.c, 463: Hi ha hagut
                                    error en escriure un bloc.\n");
                              return -1;
                        }
            case 0:     i = buf[blogicoAux];
      }

      if(nivel == 0){
            return i;
      }
      else traducir_recursivo(in, nivel-1, i, blogico % j, bfisico, reservar);
}
```

```
int traducir_bloque_inodo(unsigned int ninodo, unsigned int blogico,
      unsigned int *bfisico, char reservar){

      /* Funció principal que donat un Inodo i un BloqueLógico d'aquest,
       * es retorna el BlocFísic que es correspon.
       * Es retorna el nombre per la
       * variable 'bfisico', per complir amb el requeriment. */

      int nivel = -1;

      /* Llegim l'Inodo que ens fa falta per traduir. */
      struct Inodo in;
      in = leer_inodo(ninodo);

      /* Calcularem el nivell de profunditat del Bloc Lògic. */
      int inicio = 12;
      while(blogico >= inicio){
            nivel++;
            inicio += pow(((BLOCKSIZE / sizeof(unsigned int))), (nivel+1));
      }
      int i = 0;

      /* Implementació d'una Matriu d'Opcions:
       *          Fa que el procés de tria dels casos sigui més
       *          eficient i més ordenat que escriure sentències 'if'
       *          repetides.
       *
       *          Es pot observar com, amb només dues sentències 'if' i
       *          una sentència 'switch', s'han direccionat tots els
       *          casos possibles.
       *
       *          Descripció gràfica:
       *                          Reservat
       *                        \
       *                         \ Si__|_No_|
       *          Volem        Si_|_-1_|__1_|
       *          Reservar     No_|__0_|__0_|     */

      int r = 0, e = 0;
      if(reservar == 1) r = 1;
      switch(nivel){
            case -1: if(in.punterosDirectos[blogico] != 0) e = 1;
            default: if(in.punterosIndirectos[nivel] != 0) e = 1;
      }
      int matOpciones[2][2] = {-1, 0, 1, 0};
      switch(matOpciones[r][e]){
            case -1:    break;
            case 1:     if((i = reservar_bloque()) == -1){
                              printf("ficheros_basico.c, 521: Error! No s'ha
                                    pogut reservar el bloc.\n");
                              return -1;
                        }
                        if(nivel == -1) in.punterosDirectos[blogico] = i;
                        else in.punterosIndirectos[nivel] = i;
                        in.BloquesDataOcupados++;
            case 0:     if(nivel == -1) i = in.punterosDirectos[blogico];
                        else i = in.punterosIndirectos[nivel];
      }
```

```
	if(nivel > -1){
		/* Si tenim nivells de traduccions a Punters Indirectes,
		 * s'entrarà a la traducció recursiva del Bloc Lògic. */
		int nAux;
		for(nAux = nivel; nAux > 0; nAux--){
			blogico -=
				(int) pow((BLOCKSIZE / sizeof(unsigned int)), nAux);
		}
		blogico -= puntDirectos;
		i = traducir_recursivo(&in, nivel, i, blogico, bfisico, reservar);
		/* En sortir d'aquesta recursió, dins la variable 'i'
		 * hi tindrem el punter al Bloc que s'ha traduït. */
	}
	if(reservar == 1){
		/* Si finalment havíem de reservar el bloc, hem de guardar
		 * tots els canvis fets a l'Inodo. */
		if(escribir_inodo(in, ninodo) == -1){
			printf("ficheros_basico.c, 547: Error en escriure
				l'Inodo: %i.\n", ninodo);
			return -1;
		}
	}
	*bfisico = i;
	return 0;
}
```

```
int liberar_recursivo(struct Inodo *in, unsigned int blogico,
      int pBSig, int nivel){

      /* Funció que allibera el Bloc Físic associat al Bloc Lògic que
       * es passa per paràmetre, i retorna el nombre de Blocs
       * alliberats: Pot ser 1, o més, segons els blocs de Punters.
       * Atenció: pBSig és l'acrònim de punterBlocSegüent. */

      int liberados = 0, blogicoAux;
      unsigned int buf[BLOCKSIZE/(sizeof(unsigned int))];

      if(bread(pBSig, buf) == -1){
            printf("ficheros_basico.c, 565: Error en llegir
                  el bloc: %i.\n", pBSig);
            return -1;
      }

      int j = (int)pow((BLOCKSIZE/sizeof(unsigned int)), nivel);
      blogicoAux = blogico / j;

      if(nivel == 0){
            liberar_bloque(buf[blogicoAux]);
            buf[blogicoAux] = 0;
            in->BloquesDataOcupados--;
            return liberados + 1;
      }
      else{
            liberar_recursivo(in, blogico % j, buf[blogicoAux], nivel - 1);

            unsigned int bufAux[BLOCKSIZE/(sizeof(unsigned int))];
            memset(bufAux, 0, BLOCKSIZE);

            if(memcmp(buf, bufAux, BLOCKSIZE) == 0){
                  liberar_bloque(pBSig);
                  in->BloquesDataOcupados--;
                  liberados++;
            }
      }
}
```

```
int liberar_bloques_inodo(unsigned int ninodo, unsigned int blogico){
	/* Funció que allibera els Blocs Físics associats als Blocs Lògics
	 * de l'Inodo apuntat per 'ninodo' a partir del Bloc Lògic indicat
	 * per 'blogico' fins al final del Fitxer o Carpeta. */

	/* Llegim l'Inodo i calculam el final del Fitxer. */
	struct Inodo in;
	in = leer_inodo(ninodo);

	int bFinalFichero = in.tamBytesLogicos / BLOCKSIZE;
	int bInicio = (blogico / BLOCKSIZE);
	if(blogico % BLOCKSIZE == 0) bInicio--;
	if(in.tamBytesLogicos % BLOCKSIZE == 0) bFinalFichero--;
	if(bInicio == -1) bInicio = 0;
	if(bFinalFichero == -1) bFinalFichero = 0;

	/* Fem una iteració principal, alliberant tots els blocs que
	 * s'han especificat. */
	int i = 0, bLiberados = 0;
	for(i = bFinalFichero; i > bInicio; i--){
		if(i < puntDirectos && in.punterosDirectos[i] != 0){
			liberar_bloque(in.punterosDirectos[i]);
			in.punterosDirectos[i] = 0;
			bLiberados++;
		}
		else{
			/* Calculam el nivell del Bloc Lògic a alliberar. */
			int inicio = puntDirectos, nivel = -1;
			while(blogico >= inicio){
				nivel++;
				inicio +=
				pow(((BLOCKSIZE / sizeof(unsigned int))), (nivel+1));
			}
			bLiberados += liberar_recursivo(&in, blogico - inicio,
					in.punterosIndirectos[nivel], nivel);
		}
	}

	/* Actualitzam el Tamany en BytesLògics de l'Inodo amb els
	 * Blocs Alliberats després de tot el procés. */
	in.BloquesDataOcupados -= bLiberados;

	/* Finalment, escrivim l'Inodo en el lloc corresponent. */
	if(escribir_inodo(in, ninodo) == -1){
		printf("ficheros_basico.c, 631: Error en escriure
							l'Inodo: %i.\n", ninodo);
		return -1;
	}
	return bLiberados;
}
```

```
int liberar_inodo(unsigned int ninodo){

	/* Funció que allibera un Inodo per complet:
	 *		Tant l'Inodo com tots els blocs d'aquest seran
	 *		tornats a l'estat inicial, i enllaçat a la llista
	 *		d'Inodos lliures. */

	/* Llegim el SuperBloc i l'Inodo implicat, per recuperar les
	 * dades necessàries. */
	struct SB a;
	if(bread(SBpos, &a) == -1){
		printf("ficheros_basico.c, 647: Error en llegir el SB.\n");
		return -1;
	}

	struct Inodo in;
	in = leer_inodo(ninodo);

	/* Hem de comprovar que l'Inodo no sigui lliure, perquè si no,
	 * no té sentit fer aquesta operació. */
	if(in.tipo == 'l'){
		printf("ficheros_basico.c, 657: Error en tipus d'Inodo.\n");
		return -1;
	}

	/* Canviam el tipus i els permissos (per poder borrar) de l'Inodo.*/
	in.tipo = 'l';
	in.permisos = 7;

	/* Alliberarem tots els blocs de l'Inodo. */
	liberar_bloques_inodo(ninodo, 0);

	/* Intercanvis de Punters per posar el lliberat al principi de la
	 * llista de lliures.
	 * I actualitzam la informació del SuperBloque referent als
	 * Inodos Lliures. */
	in.punterosDirectos[0] = a.PrimerInodoLibre;
	a.PrimerInodoLibre = ninodo;

	a.InodosLibres++;

	if(escribir_inodo(in, ninodo) == -1){
		printf("ficheros_basico.c, 678: Error en escriure
			l'Inodo: %i.\n", ninodo);
		return -1;
	}
	if(bwrite(SBpos, &a) == -1){
		printf("ficheros_basico.c, 682: Error en escriure el SB.\n");
		return -1;
	}
	return ninodo;
}
```

Arxiu “ficheros_basico.h”

```
#include "bloques.h"

#define SBpos 0
#define puntDirectos 12
#define puntIndirectos 3

struct SB{
	unsigned int PrimerBloqueMB;
	unsigned int UltimoBloqueMB;
	unsigned int PrimerBloqueAI;
	unsigned int UltimoBloqueAI;
	unsigned int PrimerBloqueData;
	unsigned int UltimoBloqueData;
	/* Posicions dels Primers i Darrers Blocs de parts del Dispositiu. */
	unsigned int InodoDirectorioRaiz;
	/* Punter a l'Inodo Arrel. */
	unsigned int PrimerInodoLibre;
	/* Punter al Primer Inodo Lliure.
	 * Es renovarà cada vegada que es Reservi o s'Alliberi un Inodo. */
	unsigned int BloquesLibres;
	/* Nombre de Blocs de Dades Lliures dins el dispositiu. */
	unsigned int InodosLibres;
	/* Nombre d'Inodos Lliures a l'AI. */
	unsigned int TotalBloques;
	/* Nombre de blocs totals del dispositiu. */
	unsigned int TotalInodos;
	/* Nombre total d'Inodos que hi ha a AI. */
	char padding[BLOCKSIZE-12*sizeof(unsigned int)];
	/* Padding: Un array de caràcters per a que el 'SuperBloque'
	 * ocupi exactament 1024 bytes. */
};
```

```
struct Inodo{
      unsigned int EntradasDirectorio;
      /* Quantitat d'enllaços d'entrades. */
      unsigned int tamBytesLogicos;
      /* Tamany del fitxer o directori en bytes lògics. */
      unsigned int BloquesDataOcupados;
      /* Nombre de Blocs de Dades Ocupats per l'Inodo */
      unsigned int punterosDirectos[12];
      /* Són els 12 Punters a Blocs de Dades Directes */
      unsigned int punterosIndirectos[3];
      /* Són 3 Punters a Blocs Indirectes.
       * 3 nivells:
       *          Nivell 0: Un Bloc de Punters.
       *          Nivell 1: Un Bloc de Blocs de Punters.
       *          Nivell 2: Un Bloc de Blocs de Blocs de Punters. */
      time_t atime;
      /* Darrer accés a Dades de l'Inodo (atime) */
      time_t mtime;
      /* Darrera modificació de Dades de l'Inodo (mtime) */
      time_t ctime;
      /* Darrera modificació de l'Inodo (ctime) */
      unsigned char tipo;
      /* 3 opcions: lliure (l), directori (d) o fitxer (f) */
      unsigned char permisos;
      /* Permisos possibles: Escriptura, Lectura i/o Execució*/
      unsigned char padding[128 - (10*sizeof(unsigned int) + 10*sizeof(time_t) +
        8*sizeof(unsigned char))];
       /* El padding s'ha muntat d'aquesta manera, per qüestions internes
        * del llenguatge.*/
};

int tamMB(unsigned int nbloques);
int tamAI(unsigned int TotalINODOS);
int initSB(unsigned int nbloques, unsigned int TotalINODOS);
int initMB(unsigned int nbloques);
int initAI();
int escribir_bit(unsigned int nbloque, unsigned int bit);
unsigned char leer_bit(unsigned int nbloque);
struct Inodo leer_inodo(unsigned int ninodo);
int reservar_bloque();
int traducir_bloque_inodo(unsigned int ninodo, unsigned int blogico, unsigned
int *bfisico, char reservar);
int liberar_inodo(unsigned int ninodo);
int liberar_bloques_inodo(unsigned int ninodo, unsigned int blogico);
int liberar_recursivo(struct Inodo *in, unsigned int blogico,
                  int pBSig, int nivel);
int traducir_recursivo(struct Inodo *in, int nivel,
                  int pBSig, int blogico, unsigned int *bfisico, char reservar);
```

Arxiu “ficheros.c”

```
#include "ficheros.h"

int mi_write_f(unsigned int ninodo, const void *buffer, unsigned int offset,
      unsigned int nbytes){

      /* Funció que escriu el contingut del buffer que es passi per
       * paràmetre a la posició del fitxer (offset) indicada. */

      /* Totes les variables que haurem d'emprar són aquestes.
       * Declaram un buffer, on anirem llegint i escrivint les
       * dades que pertanyin al bloc, i llegim l'Inodo en el que hem
       * d'escriure.*/

      struct Inodo in;
      unsigned int BloqueInicial = offset / BLOCKSIZE;
      unsigned int BloqueFinal = (offset + nbytes - 1) / BLOCKSIZE;
      unsigned int ByteInicial = offset % BLOCKSIZE;
      unsigned int ByteFinal = (offset + nbytes - 1) % BLOCKSIZE;
      unsigned int BloqueFisico = 0;
      int BytesEscritos = 0;
      unsigned char buf[BLOCKSIZE];

      in = leer_inodo(ninodo);

      /* Comprovam si tenim els permisos necessaris per escriure. */
      if((in.permisos & 2) != 2){
            printf("ficheros.c, 24: No es tenen permisos
                  d'escriptura a l'Inodo %i.\n", ninodo);
            return -1;
      }

      /* Cridam a la funció per trobar el Bloc Físic corresponent. */
      if(traducir_bloque_inodo(ninodo, BloqueInicial, &BloqueFisico, 1) == -1){
            printf("ficheros.c, 30: Error en Traducir_Bloque_Inodo.\n");
            return -1;
      }

      /* Llegim el Bloc resultat de l'anterior operació. */
      if(bread(BloqueFisico, buf) == -1){
            printf("ficheros.c, 36: Error en llegir
                  el bloc: %i.\n", BloqueFisico);
            return -1;
      }

      /* Comprovam que els Blocs Inici i Final, on hem d'escriure, siguin
       * el mateix. Si és així, només farà falta cridar a 'bwrite' una
       * vegada. Si no és així, l'haurem de cridar tantes vegades com
       * faci falta. */
      if(BloqueInicial == BloqueFinal){
            memcpy(buf + ByteInicial, buffer, nbytes);

            if(bwrite(BloqueFisico, buf) == -1){
                  printf("ficheros.c, 48: Error en escriure
                        el bloc: %i.\n", BloqueFisico);
                  return -1;
            }

            BytesEscritos = nbytes;
      }
```

```
else{
	/* En cas que el Bloc Inicial i el Final no siguin el mateix,
	 * primer haurem d'escriure el que ens queda al primer bloc.*/
	memcpy(buf + ByteInicial, buffer, BLOCKSIZE - ByteInicial);

	/* I escrivim aquest primer bloc.*/
	if(bwrite(BloqueFisico, buf) == -1){
		printf("ficheros.c, 61: Error en escriure
			el bloc: %i.\n", BloqueFisico);
		return -1;
	}

	memset(buf, 0, BLOCKSIZE);

	/* Augmentam degudament els Bytes que hem escrit. */
	BytesEscritos += (BLOCKSIZE - ByteInicial);

	/* I feim l'iteració entre els Blocs que falten per escriure. */
	int j = 0;
	for(j = BloqueInicial + 1; j < BloqueFinal; j++){
		if(traducir_bloque_inodo(ninodo, j, &BloqueFisico, 1) == -1){
			printf("ficheros.c, 74: Error en reservar un bloc.\n");
			return -1;
		}

		memcpy(buf, buffer + BytesEscritos, BLOCKSIZE);

		if(bwrite(BloqueFisico, buf) == -1){
			printf("ficheros.c, 81: Error en escriure
				el bloc: %i.\n", BloqueFisico);
			return -1;
		}
		BytesEscritos += BLOCKSIZE;
	}
	/* Finalment, repetim el procés per al darrer bloc de tots. */
	if(traducir_bloque_inodo(ninodo, BloqueFinal, &BloqueFisico, 1)==-1)
	{
		printf("ficheros.c, 88: Error en reservar
			un bloc (Traduir).\n");
		return -1;
	}

	if(bread(BloqueFisico, buf) == -1){
		printf("ficheros.c, 93: Error en llegir un bloc.\n");
		return -1;
	}

	memcpy(buf, buffer + BytesEscritos, ByteFinal + 1);

	if(bwrite(BloqueFisico, buf) == -1){
		printf("ficheros.c, 100: Error en escriure
			el bloc: %i.\n", BloqueFisico);
		return -1;
	}

	BytesEscritos += (ByteFinal + 1);
}
```

```
	/* Finalment, llegim l'Inodo per guardar la nova informació. */
	in = leer_inodo(ninodo);

	if(offset + nbytes > in.tamBytesLogicos){
		in.tamBytesLogicos = offset + nbytes;
	}
	in.ctime = time(NULL);
	in.mtime = time(NULL);

	if(escribir_inodo(in, ninodo) == -1){
		printf("ficheros.c, 115: Error en escriure l'Inodo: %i.\n", ninodo);
		return -1;
	}

	return BytesEscritos;
}
```

```
int mi_read_f(unsigned int ninodo, void *buffer, unsigned int offset,
      unsigned int nbytes){

      /* Funció que retorna en el buffer del paràmetre tot el contingut
       * dins el fitxer a partir de l'offset fins a offset + nbytes. */

      /* Les variables i el buffer que ens farà falta per dur a
       * terme l'operació. */
      struct Inodo in;
      unsigned int BloqueInicial = offset / BLOCKSIZE;
      unsigned int ByteInicial = offset % BLOCKSIZE;
      unsigned int BloqueFinal = (offset + nbytes - 1) / BLOCKSIZE;
      unsigned int ByteFinal = (offset + nbytes - 1) % BLOCKSIZE;
      unsigned int bFisico = 0;
      unsigned int BytesLeidos = 0;
      unsigned char buf[BLOCKSIZE];

      /* Llegim l'Inodo implicat, i miram si tenim permisos de lectura. */
      in = leer_inodo(ninodo);
      if((in.permisos & 4) != 4){
            printf("ficheros.c, 140: No es tenen permisos de
                  lectura a l'Inodo %i.\n", ninodo);
            return -1;
      }

      if((offset + nbytes) >= in.tamBytesLogicos){
            nbytes = in.tamBytesLogicos – offset;
      }
      if(nbytes <= 0 || offset >= in.tamBytesLogicos){
            nbytes = 0;
      }

      /* Cercarem el Bloc Físic que es correspon al
       * Bloc Lògic Inicial, i el llegirem. */
      if((traducir_bloque_inodo(ninodo, BloqueInicial, &bFisico, 0) != -1)){
            if(bFisico != 0){
                  if(bread(bFisico, buf) == -1){
                        printf("ficheros.c, 153: Error en
                              llegir el Bloc: %i.\n", bFisico);
                        return -1;
                  }
            }
            else memset(buf, 0, BLOCKSIZE);
      }
```

```
	/* Si els blocs són els mateixos, copiarem del Bloc llegit.
	 * Si no, haurem de fer una iteració. */
	if(BloqueInicial == BloqueFinal){
		memcpy(buffer, buf + ByteInicial, nbytes);
		BytesLeidos = nbytes;
	}
	else{
		memcpy(buffer, buf + ByteInicial, BLOCKSIZE - ByteInicial);
		BytesLeidos = BLOCKSIZE - ByteInicial;

		int j = 0;
		for(j = BloqueInicial + 1; j < BloqueFinal; j++){
			if(traducir_bloque_inodo(ninodo, j, &bFisico, 0) != -1){
				if(bFisico != 0){
					if(bread(bFisico, buf) == -1){
						printf("ficheros.c, 175: Error en llegir
							el Bloc: %i.\n", bFisico);
						return -1;
					}
				}
				else memset(buf, 0, BLOCKSIZE);
			}
			memcpy(buffer + BytesLeidos, buf, BLOCKSIZE);
			memset(buf, 0, BLOCKSIZE);
			BytesLeidos += BLOCKSIZE;
		}

		if(traducir_bloque_inodo(ninodo, BloqueFinal, &bFisico, 0) != -1){
			if(bFisico != 0){
				if(bread(bFisico, buf) == -1){
					printf("ficheros.c, 189: Error en llegir
						el Bloc: %i.\n", bFisico);
					return -1;
				}
			}
			else memset(buf, 0, BLOCKSIZE);
		}

		memcpy(buffer + BytesLeidos, buf, ByteFinal + 1);
		BytesLeidos += (ByteFinal + 1);
	}

	/* Actualitzam les dades referents al temps.*/
	in.atime = time(NULL);

	/* Finalment escriurem l'Inodo per actualitzar les dades. */
	if(escribir_inodo(in, ninodo) == -1){
		printf("ficheros.c, 205: Error en escriure l'Inodo: %i.\n", ninodo);
		return -1;
	}
	return BytesLeidos;
}
```

```
int mi_chmod_f(unsigned int ninodo, unsigned char permisos){
	/* Funció que canvia els permisos d'un Inodo segons els que s'han
	 * especificat per paràmetre.
	 *
	 * Es llegeix l'Inodo, es canvien els permisos, s'actualitza el
	 * temps de modificació i s'escriu. */

	struct Inodo in;
	in = leer_inodo(ninodo);

	in.permisos = permisos;

	in.ctime = time(NULL);

	escribir_inodo(in, ninodo);
}

int mi_truncar_f(unsigned int ninodo, unsigned int nbytes){
	/* Funció que crida a liberar_bloques_inodo amb un valor diferent
	 * de 0.
	 * Es miren els permisos de l'Inodo i es lliberen els blocs,
	 * s'actualitzen els temps i el tamany en bytes lògics. */

	struct Inodo in;
	in = leer_inodo(ninodo);
	if((in.permisos & 2) != 2){
		printf("ficheros.c, 236: No es tenen permisos d'escriptura.\n");
		return -1;
	}

	liberar_bloques_inodo(ninodo, nbytes);
	in = leer_inodo(ninodo);

	in.mtime = time(NULL);
	in.ctime = time(NULL);

	in.tamBytesLogicos = nbytes;

	if(escribir_inodo(in, ninodo) == -1){
		printf("ficheros.c, 249: Error en escriure l'Inodo: %i.\n", ninodo);
		return -1;
	}
}

int mi_stat_f(unsigned int ninodo, struct STAT *p_stat){
	/* Funció que retorna els 'STAT' d'un Inodo. */

	struct Inodo in;
	in = leer_inodo(ninodo);

	p_stat->EntradasDirectorio = in.EntradasDirectorio;
	p_stat->tamBytesLogicos = in.tamBytesLogicos;
	p_stat->BloquesDataOcupados = in.BloquesDataOcupados;
	p_stat->ultimoAccesoDatos = in.atime;
	p_stat->ultimaModificacionDatos = in.mtime;
	p_stat->ultimaModificacionINODO = in.ctime;
	p_stat->tipo = in.tipo;
	p_stat->permisos = in.permisos;

	return 0;
}
```

Arxiu “ficheros.h”

```
#include "ficheros_basico.h"

struct STAT{
      unsigned int EntradasDirectorio;
      /* Quantitat d'enllaços d'entrades a Carpetes */
      unsigned int tamBytesLogicos;
      /* Tamany de bytes lògics */
      unsigned int BloquesDataOcupados;
      /* Nombre de Blocs de Dades Ocupats per l'Inodo */
      time_t ultimoAccesoDatos;
      /* Darrer accés a Dades de l'Inodo (atime) */
      time_t ultimaModificacionDatos;
      /* Darrera modificació de Dades de l'Inodo (mtime) */
      time_t ultimaModificacionINODO;
      /* Darrera modificació de l'Inodo (ctime) */
      unsigned char tipo;
      /* 3 opcions: lliure (l), directori (d) o fitxer (f) */
      unsigned char permisos;
      /* Permisos possibles: Escriptura, Lectura i/o Execució*/
      unsigned char padding[128 - (3*sizeof(unsigned int) +
            3*sizeof(time_t) + 2*sizeof(unsigned char))];
};

int mi_write_f(unsigned int ninodo, const void *buffer, unsigned int offset,
unsigned int nbytes);
int mi_read_f(unsigned int ninodo, void *buffer, unsigned int offset,
                  unsigned int nbytes);
int mi_chmod_f(unsigned int ninodo, unsigned char permisos);
int mi_truncar_f(unsigned int ninodo, unsigned int nbytes);
int mi_stat_f(unsigned int ninodo, struct STAT *p_stat);
```

Arxiu “directorios.c”

```
#include "directorios.h"

struct Entrada entradaGlobal;

int extraer_camino(const char *camino, char *inicial, char *final){
	char *dir;

	/* Comprovació per assegurar que el camí és correcte i es podrà
	 * fer operacions amb ell. */
	if(camino[0] != '/'){
		printf("directorios.c, 11: Camí no vàlid.\n");
		return -1;
	}

	/* Passam la primera Slash. */
	camino++;

	/* A 'dir' es trobarà la posició de la primera
	 * barra dins 'camino'.*/
	dir = strchr(camino, '/');

	/* Si 'dir' és diferent de NULL vol dir que hi ha una Slash,
	 * i per tant, vol dir que estam davant un Directori.
	 * Si no és així, som davant un fitxer. */
	if((dir-camino) < strlen(camino)){
		/* Copiam la part de 'camino' que correspon a 'inicial'. */
		strncpy(inicial, camino, (dir-camino)+1);
		inicial[(dir-camino)+1]='\0';
		/* Passam la part copiada a 'inicio'. I copiam a 'final'. */
		camino += dir-camino;
		strcpy(final, camino);
		return 0;
	}else{
		/* Copiam tot a 'inicial', i retornam que és un Fitxer. */
		strcpy(inicial, camino);
		strcpy(final, "");
		return 1;
	}
}
```

```
int buscar_entrada(const char *camino_parcial, unsigned int *p_inodo_dir,
      unsigned int *p_inodo, unsigned int *p_entrada,
      char reservar, unsigned char modo){

      /* Definició i Inicialització de Variables que Emprarem. */
      struct Inodo in;
      struct Entrada en;
      int BloquesFich = 0, BloquesLeidos = 0;
      int nEntradasFich = 0, nEntradasLeidas = 0, nEntradasFinal = 0;

      char inicial[60], final[500];
      memset(inicial, 0, 60);
      memset(final, 0, 500);

      int tipo;
      unsigned char tipoaux;

      /* Comprovam que sigui el Directori Arrel.
       * Si ho és, el retornam, si no, seguim amb l'execució. */
      if(strcmp(camino_parcial, "/") == 0){
            *p_inodo = 0;
            *p_entrada = 0;
            return 0;
      }

      /* Extreurem el camí que necessitam per fer la següent crida. */
      if((tipo = extraer_camino(camino_parcial, inicial, final)) == -1){
            printf("directorios.c, 66: Error tipus (-1):
                  Error en Extraer_Camino.\n");
            return -1;
      }
      if(tipo == 1) tipoaux = 'f';
      else if(tipo == 0) tipoaux = 'd';

      /* Inicialitzam l'Entrada. */
      memset(en.nombre, 0, 60);
      en.ninodo = -1;

      /*Llegim l'Inodo del Directori que està en curs. */
      in = leer_inodo(*p_inodo_dir);

      /* Calculam els límits dels nostres bucles. */
      BloquesFich = in.tamBytesLogicos / BLOCKSIZE;
      if((in.tamBytesLogicos % BLOCKSIZE) != 0) BloquesFich++;
      nEntradasFich = in.tamBytesLogicos / (sizeof(struct Entrada));

      struct Entrada bufAux[BLOCKSIZE/(sizeof(struct Entrada))];
      memset(bufAux, 0, BLOCKSIZE);
```

```
/* Per a Llegir Entrades per Blocs (més eficient):
 * Bucle per trobar Entrades a partir de Blocs:
 *    Mentre hi hagi Blocs per Llegir, i no s'hagi trobat l'entrada:
 *     |    Llegim Bloc;
 *     |    Mentre hi hagi Entrades per Llegir i !(hagi trobat entrada):
 *     |     |    Llegim Entrada;
 *     |    fMentre;
 *    fMentre;
 */


while((BloquesLeidos < BloquesFich) && (strcmp(inicial, en.nombre) != 0)){
      if(mi_read_f(*p_inodo_dir, bufAux,
            BloquesLeidos*BLOCKSIZE, BLOCKSIZE) == -1){

            printf("directorios.c, 98: Error tipus (-2):
                  Error en llegir Entrades.\n");
            return -2;
      }
      nEntradasLeidas = 0;
      while((nEntradasFinal < nEntradasFich) &&
            (nEntradasLeidas < (BLOCKSIZE / sizeof(struct Entrada))) &&
            (strcmp(inicial, en.nombre) != 0)){

            en = bufAux[nEntradasLeidas];
            nEntradasLeidas++;
            nEntradasFinal++;
      }
      BloquesLeidos++;
      memset(bufAux, 0, BLOCKSIZE);
}
nEntradasFinal--;

/* Al finalitzar aquest bucle, sabrem:
 *          Si nEntradasFinal == nEntradasFich(-1): hem sortit perquè
 *          no haurem trobat cap entrada amb aquest Nom a l'Inodo.
 *
 *          Si strcmp(inicial, en.nombre) == 0 voldrà dir que
 *          l'hem trobada. */

/* Si no hem trobat l'Entrada que cercàvem i som al final,
 * anirem a observar el 'reservar':
 *          Si és 0 (mode Consulta):
 *                Retornarem un error: No existeix la Entrada.
 *          Si és 1 (mode Reserva):
 *                Reservarem l'Inodo segons si és de tipus
 *                Directori o Fitxer.
 * Llavors, escriurem l'Entrada en el fitxer.
 *
 * Sino, si no hem trobat l'Entrada, no haurem trobat
 * el directori Intermig. */
```

```
	if(strcmp(inicial, en.nombre) != 0 &&
		((strcmp(final, "") == 0 || strcmp(final, "/") == 0))){
		switch(reservar){
			case 0: printf("directorios.c, 129: Error tipus (-3):
					No existeix Entrada.\n");
				return -3;
				break;
			case 1: strcpy(en.nombre, inicial);
					if((en.ninodo =
						reservar_inodo(tipoaux, modo)) == -1){

						printf("directorios.c, 134: Error en
							reservar un Inodo.\n");
						return -1;
					}
					nEntradasFinal++;
					if(mi_write_f(*p_inodo_dir, &en,
						(nEntradasFinal)*sizeof(struct Entrada),
						sizeof(struct Entrada)) == -1){

						if(en.ninodo != -1){
							liberar_inodo(en.ninodo);
						}
						printf("directorio.c, 140: Error tipus (-4):
							Error en escriure una Entrada.\n");
						return -4;
					}
					break;
		}
	}
	else if(strcmp(inicial, en.nombre) != 0 &&
		((strcmp(final, "") != 0 || strcmp(final, "/") != 0))){

		printf("directorios.c, 147: Error tipus (-6): No existeix el
			Directori Intermig = %s.\n", inicial);
		return -6;
	}

	else if(strcmp(inicial, en.nombre) == 0 &&
		((((strcmp(final, "") == 0 || strcmp(final, "/") == 0)))) &&
		(reservar == 1)){
		return -5;
	}

	/* Si hem arribat al final, tallam la recursió. */
	if((strcmp(final, "") == 0 || strcmp(final, "/") == 0)){
		*p_inodo = en.ninodo;
		*p_entrada = nEntradasFinal;
		return 0;
	}
	else{
		*p_inodo_dir = en.ninodo;
		return buscar_entrada(final, p_inodo_dir, p_inodo,
					p_entrada, reservar, modo);
	}
}



int mi_creat(char *camino, unsigned char permisos){

	/* Funció que crida a 'buscar_entrada' per crear un directori
```

```
	 * o un fitxer, especifict a camino. */
	/* Inicialitzam variables. */

	unsigned int p_inodo_dir = 0;
	unsigned int p_inodo = 0;
	unsigned int p_entrada = 0;

	/* Afegim els controls per entrar a la secció crítica. */
	mi_waitSem();
	/* Fem la crida. Els errors estan resolts a la capa superior. */
	int en = buscar_entrada(camino, &p_inodo_dir, &p_inodo,
				&p_entrada, 1, permisos);

	memcpy(entradaGlobal.nombre, camino, 60);
	entradaGlobal.ninodo = p_inodo;

	mi_signalSem();
	return en;
}

int mi_dir(char *camino, char *buffer){

	/* Funció que imprimeix el contingut del directori que es posa a
	 * camino. */
	/* Inicialitzam variables. */

	unsigned int p_inodo_dir = 0;
	unsigned int p_inodo = 0;
	unsigned int p_entrada = 0;

	/* Cridam a 'buscar_entrada'.
	 * En aquest cas hem de resoldre en el mateix mètode els errors,
	 * perquè no podem continuar l'execució amb un error d'aquests. */
	switch(buscar_entrada(camino, &p_inodo_dir, &p_inodo, &p_entrada, 0, 0)){
		case -1: printf("directorios.c, 199: Error en Extraer Camino.\n");
			 return -1; break;
		case -2: printf("directorios.c, 200: Error en Llegir Entrada.\n");
			 return -2; break;
		case -3: printf("directorios.c, 201: Error: No existeix
				 l'Entrada.\n");
			 return -3; break;
		case -4: printf("directorios.c, 202: Error en Escriure Entrada.\n");
			 return -4; break;
		case -5: printf("directorios.c, 203: Error: Entrada ja
				 existent.\n");
			 return -5; break;
		case -6: printf("directorios.c, 204: Error en trobar Directori
				 Intermig.\n");
			 return -6; break;
	}

	/* Llegim l'Inodo i miram si és un Directori i té permisos de lectura. */
	struct Inodo in;
	in = leer_inodo(p_inodo);

	if(in.tipo == 'f' || in.permisos & 2 != 2){
		printf("directorios.c, 212: Errors en el tipus o els permisos.\n");
		return -1;
	}

	/* Calculam els límits dels nostres Bucles. */
	int BloquesFichero = in.tamBytesLogicos / BLOCKSIZE;
```

```
        int BloquesLeidos = 0;
        int EntradasFichero = in.tamBytesLogicos / sizeof(struct Entrada);
        int EntradasLeidas = 0, EntradasFinal = 0;

        struct Entrada entrada;
        struct Entrada bAux[BLOCKSIZE / sizeof(struct Entrada)];

        /* Llegim totes les entrades que té el nostre Inodo en particular. */
        for(BloquesLeidos = 0; BloquesLeidos <= BloquesFichero; BloquesLeidos++){
              if(mi_read_f(p_inodo, bAux,
                           BloquesLeidos*BLOCKSIZE, BLOCKSIZE) == -1){
                    printf("directorios.c, 238: Error en llegir entrada.\n");
                    return -1;
              }

              while(EntradasFinal < EntradasFichero &&
                    EntradasLeidas < (BLOCKSIZE/sizeof(struct Entrada))){

                    entrada = bAux[EntradasLeidas];
                    in = leer_inodo(entrada.ninodo);
                    if((in.permisos & 4) == 4) strcat(buffer, "r");
                    else strcat(buffer, "-");
                    if((in.permisos & 2) == 2) strcat(buffer, "w");
                    else strcat(buffer, "-");
                    if((in.permisos & 1) == 1) strcat(buffer, "x");
                    else strcat(buffer, "-");
                    strcat(buffer, "\t");

                    char nom[60];
                    sprintf(nom, "%-15s", entrada.nombre);
                    strcat(buffer, nom);
                    strcat(buffer, "\t");

                    char tam[20];
                    sprintf(tam, "%-15i", in.tamBytesLogicos);
                    strcat(buffer, tam);

                    struct tm *tm;
                    char tmp[100];
                    tm = localtime(&in.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);

                    strcat(buffer, "|");
                    EntradasLeidas++;
                    EntradasFinal++;
              }
              EntradasLeidas = 0;
        }
}
```

```
int mi_link(char *caminoOrigen, char *caminoDest){
      /* Inicialització de Variables. */
      unsigned int p_inodo_dir = 0;
      unsigned int p_inodo = 0;
      unsigned int p_entrada = 0;
      int en = 0;

      mi_waitSem();
      /*Cridam a 'buscar_entrada'. */
      if(en = buscar_entrada(caminoOrigen, &p_inodo_dir, &p_inodo,
                                                  &p_entrada, 0, 0) < 0){
            switch(en){
                  case -1: printf("directorios.c, 287:
                              Error en Extraer Camino.\n"); break;
                  case -2: printf("directorios.c, 288:
                              Error en Llegir Entrada.\n"); break;
                  case -3: printf("directorios.c, 289:
                              Error: No existeix l'Entrada.\n"); break;
                  case -4: printf("directorios.c, 290:
                              Error en Escriure Entrada.\n"); break;
                  case -5: printf("directorios.c, 291:
                              Error: Entrada ja existent.\n"); break;
                  case -6: printf("directorios.c, 292:
                              Error en trobar Directori Intermig.\n"); break;
            }
            mi_signalSem();
            return en;
      }

      /* Seguirem endavant si l'Inodo és de tipus fitxer. */
      struct STAT stat;
      if((mi_stat_f(p_inodo, &stat) == -1) || (stat.tipo != 'f')){
            printf("directorios.c, 301: Error: Camí Origen
                                          no és un fitxer.\n", p_inodo);
            mi_signalSem();
            return -1;
      }

      /* Llegim l'Entrada que ens ha trobat l'anterior 'buscar_entrada'.*/
      struct Entrada entrada;
      if(mi_read_f(p_inodo_dir, &entrada, (p_entrada)*(sizeof(struct Entrada)),
                                          sizeof(struct Entrada)) == -1){
            printf("directorios.c, 309: Error en llegir una entrada.\n");
            mi_signalSem();
            return -1;
      }

      /* Guardam l'apuntador a l'Inodo que ens ha retornat, per poder
       * després linkar. */
      int ninodo = entrada.ninodo;
```

```
/* Inicialitzam una altra vegada per poder començar des del
 * principi. I cridam a 'buscar_entrada' per a que ens crei una
 * nova entrada.*/
p_inodo_dir = 0;
p_inodo = 0;
p_entrada = 0;

en = buscar_entrada(caminoDest, &p_inodo_dir, &p_inodo, &p_entrada, 1, 6);
if(en < 0){
      switch(en){
            case -1: printf("directorios.c, 327:
                        Error en Extraer Camino.\n"); break;
            case -2: printf("directorios.c, 328:
                        Error en Llegir Entrada.\n"); break;
            case -3: printf("directorios.c, 329:
                        Error: No existeix l'Entrada.\n"); break;
            case -4: printf("directorios.c, 330:
                        Error en Escriure Entrada.\n"); break;
            case -5: printf("directorios.c, 331:
                        Error: Entrada ja existent.\n"); break;
            case -6: printf("directorios.c, 332:
                        Error en trobar Directori Intermig.\n"); break;
      }
      mi_signalSem();
      return en;
}

/* Seguirem endavant si l'Inodo és de tipus fitxer. */
if((mi_stat_f(p_inodo, &stat) == -1) || (stat.tipo != 'f')){
      printf("directorios.c, 340: Error: Camí Destí no és un fitxer.\n");
      mi_signalSem();
      return -1;
}

if(mi_read_f(p_inodo_dir, &entrada, (p_entrada)*sizeof(struct Entrada),
                                          sizeof(struct Entrada)) == -1){
      printf("directorios.c, 346: Error en llegir una entrada.\n");
      mi_signalSem();
      return -1;
}

/* Alliberarem l'Inodo que s'ha reservat en la segona crida de
 * 'buscar_entrada' perquè només necessitavem l'entrada. */
if(liberar_inodo(entrada.ninodo) == -1){
      printf("directorios.c, 354: Error en alliberar
                                    l'Inodo: %i.\n", entrada.ninodo);
      mi_signalSem();
      return -1;
}
```

```
	/* Esto es el enlace a la Entrada nueva (de Camino2) desde la entrada de
	 * Camino1. */
	entrada.ninodo = ninodo;
	if(mi_write_f(p_inodo_dir, &entrada, (p_entrada)*sizeof(struct Entrada),
	                                        sizeof(struct Entrada)) == -1){
		printf("directorios.c, 362: Error en escriure una Entrada.\n");
		mi_signalSem();
		return -1;
	}

	struct Inodo in;
	in = leer_inodo(ninodo);
	in.EntradasDirectorio++;
	in.ctime = time(NULL);
	if(escribir_inodo(in, ninodo) == -1){
		printf("directorios.c, 372: Error en escriure
	                                        l'Inodo: %i.\n", ninodo);
		mi_signalSem();
		return -1;
	}
	mi_signalSem();
	return 0;
}
```

```
int mi_unlink(char *camino){
      /* Funció que esborra l'entrada a un directori. Si és la darrera
       * entrada, aquest directori s'ha d'eliminar. */
      unsigned int p_inodo_dir = 0;
      unsigned int p_inodo = 0;
      unsigned int p_entrada = 0;
      int en;

      mi_waitSem();
      if(en = buscar_entrada(camino, &p_inodo_dir, &p_inodo,
                                                  &p_entrada, 0, 0) < 0){
            switch(en){
                  case -1: printf("directorios.c, 392:
                              Error en Extraer Camino.\n"); break;
                  case -2: printf("directorios.c, 393:
                              Error en Llegir Entrada.\n"); break;
                  case -3: printf("directorios.c, 394:
                              Error: No existeix l'Entrada.\n"); break;
                  case -4: printf("directorios.c, 395:
                              Error en Escriure Entrada.\n"); break;
                  case -5: printf("directorios.c, 396:
                              Error: Entrada ja existent.\n"); break;
                  case -6: printf("directorios.c, 397:
                              Error en trobar Directori Intermig.\n"); break;
            }
            mi_signalSem();
            return en;
      }
      /* Llegim l'Inodo i calculam el nombre d'entrades que té. */
      struct Entrada entrada;
      struct Inodo in;
      in = leer_inodo(p_inodo_dir);
      struct STAT stat;
      mi_stat_f(p_inodo, &stat);

      if(stat.tipo == 'd' && stat.tamBytesLogicos > 0){
            printf("directorios.c, 410: Error en esborrar un directori ple.\n");
            mi_signalSem();
            return -1;
      }

      if(p_inodo == 0){
            printf("directorios.c, 416: No es pot esborrar
                                                  el directori Arrel.\n");
            mi_signalSem();
            return -1;
      }
```

```
	/* Llavors miram si és la darrera entrada:
	 *		Si no ho és, esborrarem l'entrada.
	 *			El que feim és llegir la darrera entrada i la
	 *			col·locam al lloc de la que hem d'eliminar.
	 *		Si ho és, alliberarem només l'Entrada. */
	if(p_entrada != (in.tamBytesLogicos / (sizeof(struct Entrada)))-1){
		mi_read_f(p_inodo_dir, &entrada,
				in.tamBytesLogicos - sizeof(struct Entrada),
				sizeof(struct Entrada));
		mi_write_f(p_inodo_dir, &entrada,
				p_entrada*sizeof(struct Entrada),
				sizeof(struct Entrada));
	}
	mi_truncar_f(p_inodo_dir, in.tamBytesLogicos-(sizeof(struct Entrada)));

	in = leer_inodo(p_inodo);
	in.EntradasDirectorio--;

	if(in.EntradasDirectorio <= 0) liberar_inodo(p_inodo);
	else{
		in.ctime = time(NULL);

		/* Finalment, escrivim l'Inodo si no ha estat alliberat. */
		if(escribir_inodo(in, p_inodo) == -1){
			printf("directorios.c, 440: Error en escriure
								l'Inodo: %i.\n", p_inodo);
			mi_signalSem();
			return -1;
		}
	}
	mi_signalSem();
	return 0;
}
```

```
int mi_chmod(char *camino, unsigned char permisos){

	/* Funció que canvia els permisos d'un Inodo segons els que
	 * s'especifiquen per paràmetre en aquesta funció. */

	unsigned int p_inodo_dir = 0;
	unsigned int p_inodo = 0;
	unsigned int p_entrada = 0;
	int en = 0;

	if((strcmp(entradaGlobal.nombre, camino)) == 0){
		p_inodo = entradaGlobal.ninodo;
	}
	else{
		if(en = buscar_entrada(camino, &p_inodo_dir,
					&p_inodo, &p_entrada, 0, 0) < 0){
			switch(en){
				case -1: printf("directorios.c, 462: Error en Extraer
					Camino.\n"); break;
				case -2: printf("directorios.c, 463: Error en Llegir
					Entrada.\n"); break;
				case -3: printf("directorios.c, 464: Error: No existeix
					l'Entrada.\n"); break;
				case -4: printf("directorios.c, 465: Error en Escriure
					Entrada.\n"); break;
				case -5: printf("directorios.c, 466: Error: Entrada ja
					existent.\n"); break;
				case -6: printf("directorios.c, 467: Error en trobar
					Directori Intermig.\n"); break;
			}
			mi_signalSem();
			return en;
		}
		memcpy(entradaGlobal.nombre, camino, 60);
		entradaGlobal.ninodo = p_inodo;
	}

	/* Cridam a la funció que modifica els permisos al nivell
	 * immediat inferior. */
	mi_chmod_f(p_inodo, permisos);
	mi_signalSem();
	return 0;
}
```

```
int mi_stat(char *camino, struct STAT *p_stat){

      /* Funció que retorna una struct STAT de l'Inodo al qual es
       * refereix un 'camino'. */

      unsigned int p_inodo_dir = 0;
      unsigned int p_inodo = 0;
      unsigned int p_entrada = 0;

      int e = 0;

      /* Anem a buscar l'Inodo en el qual hi ha l'entrada de camino. */
      switch(e = buscar_entrada(camino, &p_inodo_dir, &p_inodo,
                                                        &p_entrada, 0, 0)){
            case -1: printf("directorios.c, 492: Error en Extraer Camino.\n");
                     return e; break;
            case -2: printf("directorios.c, 493: Error en Llegir Entrada.\n");
                     return e; break;
            case -3: printf("directorios.c, 494: Error: No existeix
                              l'Entrada.\n");
                     return e; break;
            case -4: printf("directorios.c, 495: Error en Escriure Entrada.\n");
                     return e; break;
            case -5: printf("directorios.c, 496: Error: Entrada ja
                              existent.\n");
                     return e; break;
            case -6: printf("directorios.c, 497: Error en trobar Directori
                              Intermig.\n");
                     return e; break;
      }

      /* Per acabar, es crida a la funció implicada. */
      mi_stat_f(p_inodo, p_stat);
      return 0;
}
```

```
int mi_read(char *camino, void *buf, unsigned int offset, unsigned int nbytes){

	/* Funció que llegeix d'un Inodo al qual es refereix 'camino'. */

	unsigned int p_inodo_dir = 0;
	unsigned int p_inodo = 0;
	unsigned int p_entrada = 0;
	int en = 0;

	/* Anem a buscar l'Inodo en el qual hi ha l'entrada de camino. */
	if((strcmp(entradaGlobal.nombre, camino)) == 0){
		p_inodo = entradaGlobal.ninodo;
	}
	else{
		if(en = buscar_entrada(camino, &p_inodo_dir,
					&p_inodo, &p_entrada, 0, 0) < 0){
			switch(en){
				case -1: printf("directorios.c, 517: Error en Extraer
						Camino.\n");
					break;
				case -2: printf("directorios.c, 518: Error en Llegir
						Entrada.\n");
					break;
				case -3: printf("directorios.c, 519: Error: No existeix
						l'Entrada.\n");
					break;
				case -4: printf("directorios.c, 520: Error en Escriure
						Entrada.\n");
					break;
				case -5: printf("directorios.c, 521: Error: Entrada ja
						existent.\n");
					break;
				case -6: printf("directorios.c, 522: Error en trobar
						Directori Intermig.\n");
					break;
			}
			return en;
		}
		memcpy(entradaGlobal.nombre, camino, 60);
		entradaGlobal.ninodo = p_inodo;
	}

	/* Llegeix al Inodo que ha sortit producte de l'anterior operació. */
	if(mi_read_f(p_inodo, buf, offset, nbytes) == -1){
		printf("directorios.c: mi_read: 532:
				Error en llegir amb mi_read_f.\n");
		return -1;
	}
	return 0;
}
```

```
int mi_write(char *camino, const void *buf, unsigned int offset,
      unsigned int nbytes){

      /* Funció que escriu d'un Inodo al qual es refereix 'camino'. */
      unsigned int p_inodo_dir = 0;
      unsigned int p_inodo = 0;
      unsigned int p_entrada = 0;
      int en = 0;

      if(offset < 0){
            printf("directorios.c, 546: Error en l'offset: %i.\n", offset);
            return -1;
      }

      mi_waitSem();
      if((strcmp(entradaGlobal.nombre, camino)) == 0){
            p_inodo = entradaGlobal.ninodo;
      }
      else{
            if(en = buscar_entrada(camino, &p_inodo_dir,
                              &p_inodo, &p_entrada, 0, 0) < 0){
                  switch(en){
                        case -1: printf("directorios.c, 557: Error en Extraer
                                          Camino.\n");
                                 break;
                        case -2: printf("directorios.c, 558: Error en Llegir
                                          Entrada.\n");
                                 break;
                        case -3: printf("directorios.c, 559: Error: No existeix
                                          l'Entrada.\n");
                                 break;
                        case -4: printf("directorios.c, 560: Error en Escriure
                                          Entrada.\n");
                                 break;
                        case -5: printf("directorios.c, 561: Error: Entrada ja
                                          existent.\n");
                                 break;
                        case -6: printf("directorios.c, 562: Error en trobar
                                          Directori Intermig.\n");
                                 break;
                  }
                  mi_signalSem();
                  return en;
            }
            strcpy(entradaGlobal.nombre, camino);
            entradaGlobal.ninodo = p_inodo;
      }

      /* Escrivim al Inodo que ha sortit producte de l'anterior operació. */
      if(mi_write_f(p_inodo, buf, offset, nbytes) == -1){
            printf("directorios.c, 573: Error en escriure amb
                  mi_write_f a l'Inodo: %i.\n", p_inodo);
            mi_signalSem();
            return -1;
      }
      mi_signalSem();
      return 0;
}
```

Arxiu “directorios.h”

```
#include "ficheros.h"

struct Entrada{
      char nombre[60];
      unsigned int ninodo;
};

int extraer_camino(const char *camino, char *inicial, char *final);
int buscar_entrada(const char *camino_parcial, unsigned int *p_inodo_dir,
                  unsigned int *p_inodo, unsigned int *p_entrada,
                  char reservar, unsigned char modo);
int mi_creat(char *camino, unsigned char permisos);
int mi_chmod(char *camino, unsigned char permisos);
int mi_dir(char *camino, char *buffer);
int mi_link(char *caminoOrigen, char *caminoDest);
int mi_read(char *camino, void *buf, unsigned int offset, unsigned int nbytes);
int mi_stat(char *camino, struct STAT *p_stat);
int mi_unlink(char *camino);
int mi_write(char *camino, const void *buf, unsigned int offset,
                  unsigned int nbytes);
```

Arxiu “semaforo_mutex_posix.c”

```
#include "semaforo_mutex_posix.h"

sem_t *initSem() {
	sem_t *sem;

	sem = sem_open(SEM_NAME, O_CREAT, S_IRWXU, SEM_INIT_VALUE);
	if (sem == SEM_FAILED) {
		return NULL;
	}

	return sem;
}

void deleteSem() {
	sem_unlink(SEM_NAME);
}

void signalSem(sem_t *sem) {
	sem_post(sem);
}

void waitSem(sem_t *sem) {
	sem_wait(sem);
}
```

Arxiu “semaforo_mutex_posix.h”

```
#include <semaphore.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>

#define SEM_NAME "/mymutex"
#define SEM_INIT_VALUE 1

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

Arxiu “mi_mkfs.c”

```
#include "ficheros_basico.h"

int main(int argc, char **argv){
	if(argc != 3){
		printf("mi_mkfs.c, 5: Error en nombre de paràmetres.\n");
		printf("L'ús del programa és:\n\t");
		printf("./mi_mkfs [nom_del_dispositiu] [quantitat de blocs]\n");
		exit(0);
	}

	char *camino = argv[1];
	int numBloques;
	numBloques = atoi(argv[2]);

	/* Demanarem a l'usuari si està segur de fer un arxiu amb menys
	 * de 10000 blocs, ja que per a la simulació es necessiten almenys
	 * uns 9000 blocs en la majoria de casos. */
	if(numBloques < 10000){
		char c;
		printf("mi_mkfs, 20: Possiblement aquesta quantitat de blocs no
				seran suficients per fer la simulació.\n");
		printf("Voleu Continuar? [s/n]: ");
		scanf("%c", &c);
		if(c == 'n' || c == 'N') exit(0);
	}

	if(bmount(camino) == -1){
		printf("mi_mkfs, 27: Error en muntar el dispositiu: %s.\n", camino);
		return -1;
	}

	unsigned char buf[BLOCKSIZE];
	memset(buf, 0, BLOCKSIZE);

	int i;
	for(i = 0; i < numBloques; i++){
		if(bwrite(i, buf) == -1){
			printf("mi_mkfs, 37: Error en escriure el bloc: %i.\n", i);
			return -1;
		}
	}

	/* Inicialitzarem les tres parts del dispositiu. I reservarem
	 * l'Inodo Arrel. */
	initSB(numBloques, numBloques / 4);
	initMB(numBloques);
	initAI();

	reservar_inodo('d', 7);

	/* Finalment, desmuntam el dispositiu. */
	if(bumount() == -1){
		printf("mi_mkfs, 52: Error en desmuntar el dispositiu: %s.\n",
				camino);
		return -1;
	}

	printf("El dispositiu s'ha muntat correctament.\n");
	return 0;
}
```

Arxiu “mi_mkdir.c”

```
#include "directorios.h"

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

	if(argc != 4){
		printf("mi_mkdir, 6: Error en els arguments.\n");
		printf("L'ús del programa és:\n \t");
		printf("./mi_mkdir [nom_del_dispositiu] [permisos] [cami]\n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("mi_mkdir, 13: Error en obrir el
				dispositiu: %s.\n", argv[1]);
		return -1;
	}

	/* Hem de comprovar que els permisos són correctes. */
	int per = atoi(argv[2]);
	if(per > 7 || per < 0){
		printf("mi_mkdir, 20: Error en els permisos.\n");
		return -1;
	}

	int e = 0;
	if(e = mi_creat(argv[3], atoi(argv[2])) < 0){
		switch(e){
			case -1: printf("mi_mkdir, 27: Error en Extraer Camino.\n");
				 break;
			case -2: printf("mi_mkdir, 28: Error en Llegir Entrada.\n");
				 break;
			case -3: printf("mi_mkdir, 29: Error: No existeix
					l'Entrada.\n");
				 break;
			case -4: printf("mi_mkdir, 30: Error en Escriure Entrada.\n");
				 break;
			case -5: printf("mi_mkdir, 31: Error: Entrada ja
					existent.\n");
				 break;
			case -6: printf("mi_mkdir, 32: Error en trobar Directori
					Intermig.\n");
				 break;
		}
		bumount();
		return e;
	}
	bumount();
	return 0;
}
```

Arxiu “mi_stat.c”

```
#include "directorios.h"

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

      if(argc != 3){
            printf("mi_stat, 6: Error en els arguments.\n");
            printf("L'ús del programa és:\n \t");
            printf("./mi_stat [nom_del_dispositiu] [cami]\n");
            exit(0);
      }

    if(bmount(argv[1]) == -1){
            printf("mi_stat, 13: Error en obrir el dispositiu: %s.\n", argv[1]);
            return -1;
      }

    struct STAT p_stat;

    if(mi_stat(argv[2], &p_stat) < 0){
        printf("mi_stat, 20: Error en mi_stat.\n");
        return -1;
      }

      printf("\tTipus: \t\t\t\t\t");
      if(p_stat.tipo == 'd') printf("Directori\n");
      else if(p_stat.tipo == 'f') printf("Fitxer\n");

      printf("\tPermisos:  \t\t\t\t");
      if((p_stat.permisos & 4) == 4) printf("r");
      else printf("-");
      if((p_stat.permisos & 2) == 2) printf("w");
      else printf("-");
      if((p_stat.permisos & 1) == 1) printf("x");
      else printf("-");
      printf("\n");

      printf("\tNombre d'Entrades:\t\t\t%i\n", p_stat.EntradasDirectorio);
      printf("\tBlocs Ocupats:\t\t\t\t%i\n", p_stat.BloquesDataOcupados);
      printf("\tTamany en Bytes Lògics:\t\t\t%i\n", p_stat.tamBytesLogicos);

      struct tm *tm;
      tm = localtime(&p_stat.ultimoAccesoDatos);
      printf("\tDarrer Accés a les Dades:
                  \t\t%d-%02d-%02d %02d:%02d:%02d\t\n",
                  tm->tm_year+1900, tm->tm_mon+1,tm->tm_mday,tm->tm_hour,
                  tm->tm_min,tm->tm_sec);
      tm = localtime(&p_stat.ultimaModificacionDatos);
      printf("\tDarrera Modificació de les Dades:
                  \t%d-%02d-%02d %02d:%02d:%02d\t\n",tm->tm_year+1900,
                  tm->tm_mon+1,tm->tm_mday,tm->tm_hour,tm->tm_min,tm->tm_sec);
      tm = localtime(&p_stat.ultimaModificacionINODO);
      printf("\tDarrera Modificació de l'Inodo:
                  \t%d-%02d-%02d %02d:%02d:%02d\t\n\n",tm->tm_year+1900,
                  tm->tm_mon+1,tm->tm_mday,tm->tm_hour,tm->tm_min,tm->tm_sec);

      bumount();
}
```

Arxiu “mi_chmod.c”

```
#include "directorios.h"

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

	if(argc != 4){
		printf("mi_chmod, 6: Error en els arguments.\n");
		printf("L'ús del programa és:\n \t.");
		printf("/mi_chmod [nom_del_dispositiu] [permisos] [cami]\n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("mi_chmod, 13: Error en obrir el
				dispositiu: %s.\n", argv[1]);
		return -1;
	}

	/* Hem de comprovar que els permisos són correctes. */
	int per = atoi(argv[2]);
	if(per > 7 || per < 0){
		printf("mi_chmod, 20: Error en els permisos.\n");
		return -1;
	}

	if(mi_chmod(argv[3], atoi(argv[2])) < 0){
		printf("mi_chmod, 25: Error en mi_chmod.\n");
		bumount();
		return -1;
	}
	bumount();
	return 0;
}
```

Arxiu “mi_cat.c”

```
#include "directorios.h"

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

	if(argc != 3){
		printf("mi_cat, 6: Error en els arguments.\n");
		printf("L'ús del programa és:\n \t.");
		printf("/mi_cat [nom_del_dispositiu] [cami]\n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("mi_cat, 13: Error en obrir el dispositiu: %s.\n", argv[1]);
		return -1;
	}

	unsigned int p_inodo_dir = 0;
	unsigned int p_inodo = 0;
	unsigned int p_entrada = 0;

	switch(buscar_entrada(argv[2], &p_inodo_dir, &p_inodo, &p_entrada, 0, 0)){
		case -1: printf("mi_cat, 22: Error en Extraer Camino.\n"); break;
		case -2: printf("mi_cat, 23: Error en Llegir Entrada.\n"); break;
		case -3: printf("mi_cat, 24: Error: No existeix l'Entrada.\n");
			 break;
		case -4: printf("mi_cat, 25: Error en Escriure Entrada.\n"); break;
		case -5: printf("mi_cat, 26: Error: Entrada ja existent.\n"); break;
		case -6: printf("mi_cat, 27: Error en trobar Directori
				Intermig.\n"); break;
	}

	struct STAT p_stat;
	if(mi_stat_f(p_inodo, &p_stat) == -1){
		printf("mi_cat, 32: Error en obtenir un STAT.\n");
		return -1;
	}

	if(p_stat.tipo != 'f'){
	   printf("mi_cat, 37: El camí ha de referenciar un fitxer.\n");
	   return -1;
	}

	/* El que feim és un recorregut fins al final del fitxer i
	 * imprimim per pantalla tot el contingut, menys la brossa. */
	int offset = 0;
	int bytesLeidos = 0;
	unsigned char buffer[BLOCKSIZE];
	memset(buffer, 0, BLOCKSIZE);

	while(offset <= p_stat.tamBytesLogicos){
		bytesLeidos = mi_read_f(p_inodo, buffer, offset, BLOCKSIZE);
		write(1, buffer, bytesLeidos);
		offset += BLOCKSIZE;
		memset(buffer, 0, BLOCKSIZE);
	}

	bumount();
}
```

Arxiu “mi_escribir.c”

```
#include "ficheros.h"

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

	if(argc != 5){
		printf("mi_escribir, 6: Error en els arguments.\n");
		printf("L'ús del programa és:\n \t.");
		printf("./mi_escribir [nom_del_dispositiu] [offset] [cami]
				'[text per escriure]' \n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("mi_escribir, 13: Error en obrir el
				dispositiu: %s.\n", argv[1]);
		return -1;
	}

	if(mi_write(argv[3], argv[4], atoi(argv[2]), strlen(argv[4])) == -1){
		printf("mi_escribir, 18: Error en escriure.\n");
		bumount();
		return -1;
	}

	bumount();
	return 0;
}
```

Arxiu “leer_SF.c”

```
#include "directorios.h"

int main(int argc, char **argv){
      if(argc != 2){
            printf("leer_SF, 5: Error en el nombre d'arguments.\n");
            printf("L'ús és el següent:\n\t");
            printf("./leer_SF [nom_del_dispoditiu]");
            exit(0);
      }
      if(bmount(argv[1]) == -1){
            printf("leer_SF, 12: Error en obrir el dispositiu: %s.\n", argv[1]);
            return -1;
      }
      struct SB a;
      if(bread(SBpos, &a) == -1){
            printf("leer_SF, 18: Error en llegir el SuperBloque.\n");
            return -1;
      }

      printf("\t INFORMACIÓ del SUPERBLOQUE.\n");
      printf("\t Blocs del Mapa de Bits: Primer: %i; Darrer: %i.\n",
                  a.PrimerBloqueMB, a.UltimoBloqueMB);
      printf("\t Blocs de l'Array d'Inodos: Primer: %i; Darrer: %i.\n",
                  a.PrimerBloqueAI, a.UltimoBloqueAI);
      printf("\t Blocs del Bloc de Dades: Primer: %i; Darrer: %i.\n",
                  a.PrimerBloqueData, a.UltimoBloqueData);
      printf("\t Blocs de Dades Lliures: %i. Totals: %i.\n", a.BloquesLibres,
                  a.TotalBloques);
      printf("\t Nombre d'Inodos Lliures: %i. Totals: %i.\n", a.InodosLibres,
                  a.TotalInodos);
      printf("\t Inodo Directori Arrel: %i.\n", a.InodoDirectorioRaiz);
      printf("\n\t sizeof(struct Inodo) = %i.\n", sizeof(struct Inodo));
      printf("\n\t INFORMACIÓ del MAPA DE BITS.\n");
      printf("\t Hi ha Ocupats %i de %i Blocs.\n", a.BloquesLibres,
                  a.TotalBloques);
      printf("\n\t INFORMACIÓ dels INODOS Ocupats.\n");
      struct tm *ts;
      char atime[80];
      char mtime[80];
      char ctime[80];
      struct STAT stat;
      int ninodo;
      for (ninodo=0; ninodo < a.TotalInodos; ninodo++) {
            mi_stat_f(ninodo, &stat);
            if(stat.tipo != 'l'){
                  ts = localtime(&stat.ultimoAccesoDatos);
                  strftime(atime, sizeof(atime), "%a %Y-%m-%d %H:%M:%S", ts);
                  ts = localtime(&stat.ultimaModificacionDatos);
                  strftime(mtime, sizeof(mtime), "%a %Y-%m-%d %H:%M:%S", ts);
                  ts = localtime(&stat.ultimaModificacionINODO);
                  strftime(ctime, sizeof(ctime), "%a %Y-%m-%d %H:%M:%S", ts);
                  printf("\rID: %d     \tTamany: %i
                        \tATIME: %s   MTIME: %s   CTIME: %s\n", ninodo,
                        stat.tamBytesLogicos, atime, mtime, ctime);
            }
      }
      printf("\n");
      return 0;
}
```

Arxiu “mi_ls.c”

```
#include "directorios.h"

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

	if(argc != 3){
		printf("mi_ls, 6: Error en els arguments.\n");
		printf("L'ús del programa és:\n \t");
		printf("./mi_ls [nom_del_dispositiu] [cami]\n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("mi_ls, 13: Error en obrir el dispositiu: %s.\n", argv[1]);
		return -1;
	}

	unsigned char buf[BLOCKSIZE*30];
	memset(buf, 0, sizeof(buf));

	if(mi_dir(argv[2], buf) < 0){
		printf("mi_ls, 21: Error en mi_dir.\n");
		bumount();
		return -1;
	}
	bumount();

	char *p_buf = NULL;
	p_buf = strtok(buf, "|");
	while (p_buf != NULL){
		printf ("%s\n", p_buf);
		p_buf  = strtok(NULL, "|");
	}
}
```

Arxiu “mi_ln.c”

```
#include "directorios.h"

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

	if(argc != 4){
		printf("mi_ln, 6: Error en els arguments.\n");
		printf("L'ús del programa és:\n \t");
		printf("./mi_ln [nom_del_dispositiu] [enllaç_origen]
				[enllaç_destí] \n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("mi_ln, 13: Error en obrir el dispositiu: %s.\n", argv[1]);
		return -1;
	}

	if(mi_link(argv[2], argv[3]) == -1){
		bumount();
		printf("mi_ln, 19: Error en mi_ln.\n");
		return -1;
	}

	bumount();
	return 0;
}
```

Arxiu “mi_rm.c”

```
#include "directorios.h"

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

	if(argc != 3){
		printf("mi_rm, 6: Error en els arguments.\n");
		printf("L'ús del programa és:\n \t");
		printf("./mi_rm [nom_del_dispositiu] [enllaç] \n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("mi_rm, 13: Error en obrir el dispositiu: %s.\n", argv[1]);
		return -1;
	}

	if(mi_unlink(argv[2]) == -1){
		bumount();
		printf("mi_rm, 19: Error en mi_rm.\n");
		return -1;
	}

	bumount();
	return 0;
}
```

Arxiu “simulacion.c”

```
#include <sys/wait.h>
#include "simulacion.h"

unsigned int finished = 0;
unsigned char dirSim[60];

unsigned int posMax = 500000;
/*(((12+256+256*256+256*256*256)-1)*BLOCKSIZE)/sizeof(struct Registro);*/

void enterrador(){
	while(wait3(NULL, WNOHANG, NULL) > 0){
		finished++;
		printf("Processos Acabats: %i de %i.\n", finished, MAX_PROCESOS);
	}
}

void CrearSimulacion(){
	memset(dirSim, 0, 60);

	struct tm *tm;
	time_t tiempo = time(NULL);
	tm = localtime(&tiempo);
	sprintf(dirSim, "/simul_%d%02d%02d%02d%02d%02d/", tm->tm_year + 1900,
			tm->tm_mon + 1, tm->tm_mday, tm->tm_hour, tm->tm_min,
			tm->tm_sec);
	if(mi_creat(dirSim, 7) == -1){
		printf("simulacion.c, 25: Error en crear un Directori.\n");
	}
}

int CrearHijo(){
	struct Registro r;
	int i;
	char pathHijo[60];
	char p[60];
	memset(pathHijo, 0, 60);
	memset(p, 0, 60);
	sprintf(pathHijo, "%sproceso_%d/", dirSim, getpid());
	sprintf(p, "%sprueba.dat", pathHijo);
	mi_creat(pathHijo, 7);
	mi_creat(p, 7);

	int max = 0;
	srand(time(NULL) + getpid());
	r.numEsc = 0;
	for(i = 0; i < MAX_PROCESOS/2; i++){
		r.time = time(NULL);
		r.pid = getpid();
		r.numEsc = i + 1;
		r.posFich = (rand() % posMax);
		if(r.posFich > max) max = r.posFich;
		mi_write(p, &r, r.posFich*sizeof(struct Registro),
			sizeof(struct Registro));
		usleep(50000);
	}
}
```

```
int main(int argc, char **argv){
	int i;
	if(argc != 2){
		printf("simulacion.c, 58: Error en nombre de paràmetres.\n");
		printf("L'ús del programa és:\n\t");
		printf("./simulacion [nom_del_dispositiu] \n");
		return -1;
	}

	if(bmount(argv[1]) == -1){
		printf("simulacion.c, 65: Error en Muntar el Fitxer.\n");
		return -1;
	}

	CrearSimulacion();
	signal(SIGCHLD, enterrador);

	for(i = 0; i < MAX_PROCESOS; i++){
		if(fork() == 0){
			CrearHijo();
			exit(0);
		}
		usleep(200000);
	}
	while(finished < MAX_PROCESOS) pause();

	printf("Final de la Simulació.\n");
	bumount();
}
```

Arxiu “simulacion.h”

```
#include <stdio.h>
#include <sys/wait.h>
#include "directorios.h"

#define MAX_PROCESOS 100

struct Registro{
	time_t time;
	unsigned int pid;
	unsigned int numEsc;
	unsigned int posFich;
};
```

Arxiu “verificacion.c”

```
#include "simulacion.h"

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

	if(argc != 2){
		printf("verificacion.c, 6:Error en els arguments.\n");
		printf("L'ús del programa és:\n \t.");
		printf("/verificacion [nom_del_dispositiu] \n");
		exit(0);
	}

	if(bmount(argv[1]) == -1){
		printf("verificacion.c, 13: Error en obrir
				el dispositiu: %s.\n", argv[1]);
		return -1;
	}

	unsigned char dirSim[60];
	sprintf(dirSim, "/");

	struct Entrada en;
	mi_read(dirSim, &en, 0, sizeof(struct Entrada));
	sprintf(dirSim, "/%s", en.nombre);

	struct STAT stat;
	mi_stat(dirSim, &stat);

	if((stat.tamBytesLogicos / (sizeof(struct Entrada))) != MAX_PROCESOS){
		printf("verificacion.c, 28: Error en el nombre de processos.\n");
		return -1;
	}

	int i = 0, j = 0;
	int ninodo = en.ninodo;
	struct Entrada enAux;
	struct STAT statAux;
	struct Registro registros[BLOCKSIZE/sizeof(struct Registro)];

	unsigned char blanco[sizeof(struct Registro)];
	memset(blanco, 0, sizeof(struct Registro));

	int BloquesInicial = 0, BloquesFinal;
	FILE *f = fopen("informe.txt", "w");
```

```
for(i = 0; i < MAX_PROCESOS; i++){
      mi_read_f(ninodo, &en, i*(sizeof(struct Entrada)),
                  sizeof(struct Entrada));

      mi_read_f(en.ninodo, &enAux, 0, sizeof(struct Entrada));

      mi_stat_f(enAux.ninodo, &statAux);

      BloquesFinal = statAux.tamBytesLogicos / BLOCKSIZE;
      if((statAux.tamBytesLogicos % BLOCKSIZE) != 0) BloquesFinal++;

      struct Registro regMayor, regMenor, regPrincipio, regFinal;
      regMenor.posFich = UINT_MAX;
      regMayor.posFich = 0;
      regPrincipio.time = time(NULL);
      regFinal.time = time(NULL);

      char *p_pid;
      p_pid = strchr(en.nombre, '_');
      int pidAux = atoi(p_pid+1);
      int validados = 0;

      for(BloquesInicial = 0; BloquesInicial < BloquesFinal;
                                          BloquesInicial++){
            mi_read_f(enAux.ninodo, registros,
                        BloquesInicial*BLOCKSIZE, BLOCKSIZE);

            for(j = 0; j < (BLOCKSIZE / sizeof(struct Registro)); j++){
                  if(memcmp(&registros[j], blanco,
                              sizeof(struct Registro)) != 0){
                        if(pidAux == registros[j].pid){
                              validados++;
                              if(registros[j].posFich < regMenor.posFich){
                                    memcpy(&regMenor, &registros[j],
                                                sizeof(struct Registro));
                              }
                              if(registros[j].posFich > regMayor.posFich){
                                    memcpy(&regMayor, &registros[j],
                                                sizeof(struct Registro));
                              }
                              if(registros[j].time < regPrincipio.time){
                                    memcpy(&regPrincipio, &registros[j],
                                                sizeof(struct Registro));
                              }
                              if(registros[j].time > regFinal.time){
                                    memcpy(&regFinal, &registros[j],
                                                sizeof(struct Registro));
                              }
                        }
                  }
            }
      }
```

```
            char tiempo[60];
            char cadena[500];
            struct tm *tm;
            sprintf(cadena, "Verificació del procés %i:\n", pidAux);
            fputs(cadena, f);
            printf("%s", cadena);
            sprintf(cadena, "\tNombre de Validacions: %i.\n", validados);
            fputs(cadena, f);
            printf("%s", cadena);
            sprintf(cadena, "\tInformació dels Registres:\n");
            fputs(cadena, f);
            printf("%s", cadena);
            sprintf(cadena, "\t\tPosició menor = %i.\n", regMenor.posFich);
            fputs(cadena, f);
            printf("%s", cadena);
            sprintf(cadena, "\t\tPosició major = %i.\n", regMayor.posFich);
            fputs(cadena, f);
            printf("%s", cadena);
            tm = localtime(&regPrincipio.time);
            sprintf(tiempo, "%d-%02d-%02d %02d:%02d:%02d",
                    tm->tm_year + 1900,
                    tm->tm_mon + 1,
                    tm->tm_mday,
                    tm->tm_hour,
                    tm->tm_min,
                    tm->tm_sec);
            sprintf(cadena, "\t\tTemps primer = %s\n", tiempo);
            fputs(cadena, f);
            printf("%s", cadena);
            tm = localtime(&regFinal.time);
            sprintf(tiempo, "%d-%02d-%02d %02d:%02d:%02d",
                    tm->tm_year + 1900,
                    tm->tm_mon + 1,
                    tm->tm_mday,
                    tm->tm_hour,
                    tm->tm_min,
                    tm->tm_sec);
            sprintf(cadena, "\t\tTemps darrer = %s\n", tiempo);
            fputs(cadena, f);
            printf("%s", cadena);
        }
        fclose(f);
}
```

Arxiu “script1.sh”

```
#!/bin/bash

# $1 --> Disco virtual

# Intentarem llistar el dispositiu buit
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Crearem un directori dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori1/-------
echo
./mi_mkdir $1 7 /directori1/

# Tornarem llistar el contingut del dispositiu i veurem que s'ha creat
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Crearem una sèrie de directoris i fitxers fins arribar al
# límit d'un bloc.
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori2/-------
echo
./mi_mkdir $1 7 /directori2/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /fitxer3-----------
echo
./mi_mkdir $1 7 /fitxer3

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori4/-------
echo
./mi_mkdir $1 7 /directori4/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /fitxer5-----------
echo
./mi_mkdir $1 7 /fitxer5

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori6/-------
echo
./mi_mkdir $1 7 /directori6/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /fitxer7-----------
echo
./mi_mkdir $1 7 /fitxer7

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori8/-------
echo
./mi_mkdir $1 7 /directori8/
```

```
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori9/-------
echo
./mi_mkdir $1 7 /directori9/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori10/-------
echo
./mi_mkdir $1 7 /directori10/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori11/-------
echo
./mi_mkdir $1 7 /directori11/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori12/-------
echo
./mi_mkdir $1 7 /directori12/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori13/-------
echo
./mi_mkdir $1 7 /directori13/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori14/-------
echo
./mi_mkdir $1 7 /directori14/

echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori15/-------
echo
./mi_mkdir $1 7 /directori15/

# Llistarem el contingut del dispositiu i veurem què s'ha creat
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Mostrarem les dades de l'Arrel del dispositiu, comprovant que
# encara només té ocupat un sol Blocs de Dades.
echo
echo MI_STAT -- ./mi_stat $1 /---------------------
echo
./mi_stat $1 /

# Crearem, per mitjà de mi_ln, un link nou a un fitxer nou.
echo
echo MI_LN -- ./mi_ln $1 /fitxer3 /fitxer16--------
echo
./mi_ln $1 /fitxer3 /fitxer16

# Tornarem a mostrar les dades de l'Arrel del dispositiu per veure
# que ara ocupa un Bloc de Dades més.
echo
echo MI_STAT -- ./mi_stat $1 /---------------------
echo
./mi_stat $1 /
```

```
# Llistarem el contingut del dispositiu i veurem què s'ha creat
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Eliminarem un directori de l'Arrel del dispositiu.
echo
echo MI_RM -- ./mi_rm $1 /directori14/
echo
./mi_rm $1 /directori14/

# Llistarem el contingut del dispositiu i veurem què s'ha creat
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Tornarem a mostrar les dades de l'Arrel del dispositiu per veure
# que ara ocupa un Bloc de Dades menys.
echo
echo MI_STAT -- ./mi_stat $1 /---------------------
echo
./mi_stat $1 /

# Eliminarem un altre directori de l'Arrel del dispositiu.
echo
echo MI_RM -- ./mi_rm $1 /directori10/
echo
./mi_rm $1 /directori10/

# Llistarem el contingut del dispositiu
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Tornarem a mostrar les dades de l'Arrel del dispositiu per veure
# que ara ocupa menys Bytes.
echo
echo MI_STAT -- ./mi_stat $1 /---------------------
echo
./mi_stat $1 /
```

Arxiu “script2.sh”

```
#!/bin/bash

# $1 --> Dispositiu Virtual

# Intentarem llistar el dispositiu buit
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Crearem un directori dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori/-------
echo
./mi_mkdir $1 7 /directori/

# Tornarem llistar el contingut del dispositiu i veurem que s'ha creat
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Llistarem el contingut del directori creat, que serà buit.
echo
echo MI_LS -- ./mi_ls $1 /directori/---------------
echo
./mi_ls $1 /directori/

# Crearem un fitxer dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /fitxer-----------
echo
./mi_mkdir $1 7 /fitxer

# Tornarem a llistar el contingut del dipositiu i veurem que s'ha creat el
fitxer.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Escriurem un contingut al fitxer creat que ocupi més d'un bloc.
echo
echo MI_ESCRIBIR -- ./mi_escribir $1 25120 /fitxer-
echo
./mi_escribir $1 25120 /fitxer "Aixo es una prova d'escriptura de text que
ocupa mes o menys 500 caracters, i per tant, s'escriu en blocs logics diferents,
perque 25*1024 = 25600. Si es vol escriure mes text, es pot escriure aqui.
Si se'n vol menys, s'haura de borrar. Es pot provar d'escriure menys text,
pero es un exemple poc il·lustratiu. S'escriu sense accents perque es
possible que doni errors amb la codificacio del fitxer de text, perque no
s'obre amb naturalitat amb 'gedit' i per aixo emprarem 'Okteta'. Creat
dia 04-08-2013."

# Mostrarem el tamany del fitxer anterior.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /
```

```
# Llegirem el contingut del fitxer, i es direccionarà la sortida a un
# fitxer. Ara es pot comprovar que són igual de grans.
# Es pot obrir amb l'editor 'Okteta', perquè amb 'gedit' no respon.
echo
echo MI_CAT -- ./mi_cat $1 /fitxer \> sortidaMiCat--
echo
./mi_cat $1 /fitxer > sortidaMiCat
echo
```

Arxiu “script3.sh”

```
#!/bin/bash

# $1 --> Dispositiu Virtual

#Llistarem el contingut de l'Arrel creat, que serà buit.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Crearem un fitxer dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /fitxer-----------
echo
./mi_mkdir $1 7 /fitxer

# Tornarem a llistar el contingut del dipositiu i veurem que s'ha creat el
fitxer.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Escriurem un contingut al fitxer creat que ocupi més d'un bloc.
echo
echo MI_ESCRIBIR --./mi_escribir $1 25120 /fitxer-
echo
./mi_escribir $1 25120 /fitxer "Escrit a partir de l'Offset 25120."

# Tornarem a llistar el contingut del dipositiu i veurem que s'ha omplit el
fitxer.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Escriurem un contingut al fitxer creat que ocupi més d'un bloc.
echo
echo MI_ESCRIBIR --./mi_escribir $1 265000 /fitxer-
echo
./mi_escribir $1 256000 /fitxer "Escrit a partir de l'Offset 256000."

# Tornarem a llistar el contingut del dipositiu i veurem que s'ha omplit el
fitxer.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Escriurem un contingut al fitxer creat que ocupi més d'un bloc.
echo
echo MI_ESCRIBIR --./mi_escribir $1 30720000 /fitxer-
echo
./mi_escribir $1 30720000 /fitxer "Escrit a partir de l'Offset 30720000."
```

```
# Tornarem a llistar el contingut del dipositiu i veurem que s'ha omplit el
fitxer.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Escriurem un contingut al fitxer creat que ocupi més d'un bloc.
echo
echo MI_ESCRIBIR --./mi_escribir $1 71680000 /fitxer-
echo
./mi_escribir $1 71680000 /fitxer "Escrit a partir de l'Offset 71680000."

# Tornarem a llistar el contingut del dipositiu i veurem que s'ha omplit el
fitxer.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /
```

Arxiu “script4.sh”

```
#!/bin/bash

# $1 --> Dispositiu Virtual

#Llistarem el contingut de l'Arrel creat, que serà buit.
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Crearem un fitxer dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /fitxer-----------
echo
./mi_mkdir $1 7 /fitxer

# Crearem un directori dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori/-------
echo
./mi_mkdir $1 7 /directori/

# Tornarem llistar el contingut del dispositiu i veurem que s'ha creat
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /

# Canviarem els permisos dels dos
echo
echo MI_CHMOD -- ./mi_chmod $1 4 /directori/
echo
./mi_chmod $1 4 /directori/

echo
echo MI_CHMOD -- ./mi_chmod $1 1 /fitxer
echo
./mi_chmod $1 1 /fitxer

# Tornarem llistar el contingut del dispositiu
echo
echo MI_LS -- ./mi_ls $1 /-------------------------
echo
./mi_ls $1 /
```

Arxiu “script5.sh”

```
#!/bin/bash

# $1 --> Dispositiu Virtual

#Llistarem el contingut de l'Arrel creat, que serà buit.
echo
echo MI_LS -- ./mi_ls $1 /------------------------
echo
./mi_ls $1 /

# Crearem un fitxer dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori/--------
echo
./mi_mkdir $1 7 /directori/

# Crearem un fitxer dins l'Arrel
echo
echo MI_MKDIR -- ./mi_mkdir $1 7 /directori/fitxer--------
echo
./mi_mkdir $1 7 /directori/fitxer

# Llistarem el contingut del directori creat.
echo
echo MI_LS -- ./mi_ls $1 /directori/---------------
echo
./mi_ls $1 /directori/

# Crearem, per mitjà de mi_ln, un link nou a un fitxer nou.
echo
echo MI_LN -- ./mi_ln $1 /directori/fitxer /fitxer--------
echo
./mi_ln $1 /directori/fitxer /fitxer

# Escriurem un contingut al fitxer.
echo
echo MI_ESCRIBIR --./mi_escribir $1 25120 /directori/fitxer-
echo
./mi_escribir $1 25120 /directori/fitxer "Escrit a partir de l'Offset 25120."

# Llistarem el contingut del directori creat, que serà buit.
echo
echo MI_LS -- ./mi_ls $1 /---------------
echo
./mi_ls $1 /
```