Tiposdatos.h

Escribir esta estructura con los datos en el sector de arranque.

struct SECBOOT {

char jump[4];

char nombre\_disco[8];

unsigned char sec\_res; // 1 sector de arranque

unsigned char sec\_mapa\_bits\_nodos\_i; // 1 sector

unsigned char sec\_mapa\_bits\_bloques; // 4 sectores

unsigned short sec\_tabla\_nodos\_i; // 1 sector

unsigned short sec\_log\_unidad; // 54400 sectores

unsigned char sec\_x\_bloque; // 4 sectores por bloque

unsigned char heads; // 20 superficies

unsigned char cyls; // 160 cilindros

unsigned char secfis; // 17 sectores

char restante[487];

};

Formatear el disco consiste en:

1. Escribir el sector de arranque con los parámetros del disco en el sector físico 1, cilindro 0, superficie 0.
2. Reiniciar los mapas de bits (todos en 0)
3. Reiniciar la tabla de nodos i.

Estructura de datos que vamos a usar para los nodos i.

struct INODE {

char name[20];

unsigned short uid;

unsigned short gid;

unsigned short perms;

unsigned int datetimecreat;

unsigned int datetimemodif;

unsigned int size;

unsigned short blocks[10];

unsigned short indirect;

unsigned short indirect2;

};

Funciones para el manejo de los mapas de bits (area de datos)

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Para el mapa de bits del área de de datos

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int isblockfree(int block)

{

int offset=block/8; // Obtener en que byte

int shift=block%8; // y bit, que indica si el bloque está libte

int result;

int i;

// Es importante tenerlo en memoria porque ahí es donde tenemos

// la información del disco, cuáles son los sectores donde hay qué

if(!secboot\_en\_memoria)

{

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

// Calculo el sector donde está el mapa de bits para los bloques

mapa\_bits\_bloques= secboot.sec\_res+secboot.sec\_mapa\_bits\_nodos\_i;

// ¿Está en memoria el mapa de bits de bloques?, si no, cargarlo a // memoria

if(!blocksmap\_en\_memoria)

{

// Leer todos los sectores del mapa de bits a memoria

for(i=0;i<secboot.sec\_mapa\_bits\_bloques;i++)

result=**vdreadseclog**(mapa\_bits\_bloques+i,blocksmap+i\*512);

blocksmap\_en\_memoria=1;

}

if(blocksmap[offset] & (1<<shift))

return(0);

else

return(1);

}

// Buscar en el mapa de bits cuál es el siguiente bloque libre

int nextfreeblock()

{

int i,j;

int result;

// Ya quedó explicado arriba

if(!secboot\_en\_memoria)

{

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

mapa\_bits\_bloques = secboot.sec\_res+secboot.sec\_mapa\_bits\_nodos\_i;

if(!blocksmap\_en\_memoria)

{

for(i=0;i<secboot.sec\_mapa\_bits\_bloques;i++)

result=vdreadseclog(mapa\_bits\_bloques+i,blocksmap+i\*512);

blocksmap\_en\_memoria=1;

}

// Buscar el primer byte en el mapa de bloques donde hay al menos un

// bloque libre

i=0;

while(blocksmap[i]==0xFF && i<secboot.sec\_mapa\_bits\_bloques\*512)

i++;

// Si no llegamos al final

if(i<secboot.sec\_mapa\_bits\_bloques\*512)

{

j=0;

while(blocksmap[i] & (1<<j) && j<8)

j++;

return(i\*8+j); // Regresando cual es el primer bloque libre encontrado

}

else

return(-1); // Llegamos al final del mapa y no encontramos

// un bloque libre

}

Poner un bloque como no disponible

int assignblock(int block)

{

int offset=block/8;

int shift=block%8;

int result;

int i;

int offset;

if(!secboot\_en\_memoria)

{

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

mapa\_bits\_bloques= secboot.sec\_res+secboot.sec\_mapa\_bits\_nodos\_i;

if(!blocksmap\_en\_memoria)

{

for(i=0;i<secboot.sec\_mapa\_bits\_bloques;i++)

result=vdreadseclog(mapa\_bits\_bloques+i,blocksmap+i\*512);

blocksmap\_en\_memoria=1;

}

// Poner en 1 el bit en el byte que corresponde al número de bloque

blocksmap[offset]|=(1<<shift);

// Escribir ese sector en el disco

offset=(offset/512)\*512;

// 0 .. 511 = 0

// 512 .. 1023 = 512

// 1024 .. 1535 = 1024

// offset=offset-(offset%512);

vdwriteseclog(mapa\_bits\_bloques+offset,blocksmap+sector\*512);

return(1);

}

Establecer el bloque como libre

int unassignblock(int block)

{

int offset=block/8;

int shift=block%8;

int result;

char mask;

int sector;

int i;

if(!secboot\_en\_memoria)

{

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

mapa\_bits\_bloques= secboot.sec\_res+secboot.sec\_mapa\_bits\_nodos\_i;

if(!blocksmap\_en\_memoria)

{

for(i=0;i<secboot.sec\_mapa\_bits\_bloques;i++)

result=vdreadseclog(mapa\_bits\_bloques+i,blocksmap+i\*512);

blocksmap\_en\_memoria=1;

}

blocksmap[offset]&=(char) **~**(1<<shift);

sector=(offset/512)\*512;

vdwriteseclog(mapa\_bits\_bloques+sector,blocksmap+sector\*512);

return(1);

}

Funciones para manipular los mapas de bits de los nodos i

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Para el mapa de bits del área de nodos i

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int isinodefree(int inode)

{

int offset=inode/8;

int shift=inode%8;

int result;

// Checar si el sector de boot está en memoria

if(!secboot\_en\_memoria)

{

// Si no está en memoria, cárgalo

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

mapa\_bits\_nodos\_i= secboot.sec\_res; //Usamos la información del sector de boot para

//determinar en que sector inicia el

// mapa de bits de nodos i

// Está el mapa está en memoria

if(!inodesmap\_en\_memoria)

{

// Si no está en memoria, hay que leerlo del disco

result=vdreadseclog(mapa\_bits\_nodos\_i,inodesmap);

inodesmap\_en\_memoria=1;

}

if(inodesmap[offset] & (1<<shift))

return(0);

else

return(1);

}

int nextfreeinode()

{

int i,j;

int result;

if(!secboot\_en\_memoria)

{

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

mapa\_bits\_nodos\_i= secboot.sec\_res;

if(!inodesmap\_en\_memoria)

{

result=vdreadseclog(mapa\_bits\_nodos\_i,inodesmap);

inodesmap\_en\_memoria=1;

}

// Recorrer byte por byte mientras sea 0xFF sigo recorriendo

i=0;

while(inodesmap[i]==0xFF && i<secboot.sec\_mapa\_bits\_nodos\_i)

i++;

if(i<secboot.sec\_mapa\_bits\_nodos\_i)

{

j=0;

while(inodesmap[i] & (1<<j) && j<8)

j++;

return(i\*8+j);

}

else

return(-1);

}

int assigninode(int inode)

{

int offset=inode/8;

int shift=inode%8;

int result;

if(!secboot\_en\_memoria)

{

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

mapa\_bits\_nodos\_i= secboot.sec\_res;

if(!inodesmap\_en\_memoria)

{

result=vdreadseclog(mapa\_bits\_nodos\_i,inodesmap);

inodesmap\_en\_memoria=1;

}

inodesmap[offset]|=(1<<shift);

vdwriteseclog(mapa\_bits\_nodos\_i,inodesmap);

return(1);

}

int unassigninode(int inode)

{

int offset=inode/8;

int shift=inode%8;

int result;

char mask;

if(!secboot\_en\_memoria)

{

result=vdreadsector(0,0,0,1,1,(char \*) &secboot);

secboot\_en\_memoria=1;

}

mapa\_bits\_nodos\_i= secboot.sec\_res;

if(!inodesmap\_en\_memoria)

{

result=vdreadseclog(mapa\_bits\_nodos\_i,inodesmap);

inodesmap\_en\_memoria=1;

}

inodesmap[offset]&=(char) ~(1<<shift);

vdwriteseclog(mapa\_bits\_nodos\_i,inodesmap);

return(1);

}