# Algunas estructuras del Sistema de archivos

Esta es una estructura de datos que vamos a usar para guardar las fechas y horas desempaquetadas.

struct DATE {

int year;

int month;

int day;

int hour;

int min;

int sec;

};

# Funciones para el manejo de los nodos i. Estas funciones usan el mapa de bits del área de nodos i para determinar si un nodo i está libre, ocupado, y también tenemos funciones para asignar y liberar nodos i

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

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

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

// Usando el mapa de bits, determinar si un nodo i, está libre u ocupado.

int isinodefree(int inode)

{

int offset=inode/8;

int shift=inode%8;

int result;

// Checar si el sector de boot de la partición está en memoria

if(!secboot\_en\_memoria)

{

// Si no está en memoria, cárgalo

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

secboot\_en\_memoria=1;

}

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

//Usamos la información del sector de boot de la partición para

//determinar en que sector inicia el

// mapa de bits de nodos i

// Ese 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); // El nodo i ya está ocupado

else

return(1); // El nodo i está libre

}

// Buscar en el mapa de bits, el primer nodo i que esté libre, es decir, que su bit está en 0, me regresa ese dato

int nextfreeinode()

{

int i,j;

int result;

// Checar si el sector de boot de la partición está en memoria

if(!secboot\_en\_memoria)

{

// Si no está en memoria, cárgalo

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

secboot\_en\_memoria=1;

}

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

//Usamos la información del sector de boot de la partición para

//determinar en que sector inicia el

// mapa de bits de nodos i

// Ese 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;

}

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

i=0;

while(inodesmap[i]==0xFF && i<3)

i++;

if(i<3)

{

// Recorrer los bits del byte, para encontrar el bit

// que está en cero

j=0;

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

j++;

return(i\*8+j); // Regresar el bit del mapa encontrado en cero

}

else // Todos los bits del mapa de nodos i están en 1

return(-1); // -1 significa que no hay nodos i disponibles

}

// Poner en 1, el bit que corresponde al número de inodo indicado

int assigninode(int inode)

{

int offset=inode/8;

int shift=inode%8;

int result;

// Checar si el sector de boot de la partición está en memoria

if(!secboot\_en\_memoria)

{

// Si no está en memoria, cárgalo

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

secboot\_en\_memoria=1;

}

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

//Usamos la información del sector de boot de la partición para

//determinar en que sector inicia el

// mapa de bits de nodos i

// Ese 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;

}

inodesmap[offset]|=(1<<shift); // Poner en 1 el bit indicado

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

return(1);

}

// Poner en 0, el bit que corresponde al número de inodo indicado

int unassigninode(int inode)

{

int offset=inode/8;

int shift=inode%8;

int result;

// Checar si el sector de boot de la partición está en memoria

if(!secboot\_en\_memoria)

{

// Si no está en memoria, cárgalo

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

secboot\_en\_memoria=1;

}

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

//Usamos la información del sector de boot de la partición para

//determinar en que sector inicia el

// mapa de bits de nodos i

// Ese 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;

}

inodesmap[offset]&=(char) ~(1<<shift); // Poner en cero el bit que corresponde al inodo indicado

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

return(1);

}

# Funciones para el manejo de los bloques del área de datos. Estas funciones usan el mapa de bits del área de datos para determinar si un bloque está libre, ocupado, y también tenemos funciones para asignar y liberar bloques.

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

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

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

// Usando la información del mapa de bits del area de datos, saber si un bloque está libre o no

int isblockfree(int block)

{

int offset=block/8; // Número de byte en el mapa

int shift=block%8; // Número de bit en el byte

int result;

int i;

// Determinar si tenemos el sector de boot de la partición en memoria

if(!secboot\_en\_memoria)

{

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

secboot\_en\_memoria=1;

}

// Calcular el sector lógico donde está el mapa de bits de los bloques

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

// Verificar si ya está en memoria, si no, cargarlo

if(!blocksmap\_en\_memoria)

{

// Cargar todos los sectores que corresponden al

// mapa de bits

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); // Si el bit está en 1, regresar 0 (no está libre)

else

return(1); // Si el bit está en 0, regresar 1 (si está libre)

}

// Usando el mapa de bits, buscar el siguiente bloque libre

int nextfreeblock()

{

int i,j;

int result;

// Determinar si tenemos el sector de boot de la partición en memoria

if(!secboot\_en\_memoria)

{

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

secboot\_en\_memoria=1;

}

// Calcular el sector lógico donde está el mapa de bits de los bloques

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

// Verificar si ya está en memoria, si no, cargarlo

if(!blocksmap\_en\_memoria)

{

// Cargar todos los sectores que corresponden al

// mapa de bits

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

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

blocksmap\_en\_memoria=1;

}

// Empezar desde el primer byte del mapa de bloques.

i=0;

// Si el byte tiene todos los bits en 1, y mientras no

// lleguemos al final del mapa de bits

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

i++;

// Si no llegué al final del mapa de bits, quiere decir

// que aún hay bloques libres

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

{

j=0;

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

j++;

return(i\*8+j); // Retorno el número de bloque

// que se encontró disponible

}

else

// Si ya no hublo bloques libres, regresar -1 = Error

return(-1);

}

// Asignar un bloque en 1 en el mapa de bits, lo cual significa que estaría ya ocupado.

int assignblock(int block)

{

int offset=block/8;

int shift=block%8;

int result;

int i;

int sector;

// Determinar si tenemos el sector de boot de la partición en memoria

if(!secboot\_en\_memoria)

{

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

secboot\_en\_memoria=1;

}

// Calcular el sector lógico donde está el mapa de bits de los bloques

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

// Verificar si ya está en memoria, si no, cargarlo

if(!blocksmap\_en\_memoria)

{

// Cargar todos los sectores que corresponden al

// mapa de bits

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]|=(1<<shift);

// Determinar en que número de sector está el bit que

// modificamos

sector=(offset/512);

// Escribir el sector del mapa de bits donde está el bit

// que modificamos

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

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

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

return(1);

}

// Poner en 0 el bit que corresponde a un bloque en el mapa de bits, esto equivale a decir que el bloque está libre

int unassignblock(int block)

{

int offset=block/8;

int shift=block%8;

int result;

char mask;

int sector;

int i;

// Determinar si tenemos el sector de boot de la partición en memoria

if(!secboot\_en\_memoria)

{

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

secboot\_en\_memoria=1;

}

// Calcular el sector lógico donde está el mapa de bits de los bloques

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

// Verificar si ya está en memoria, si no, cargarlo

if(!blocksmap\_en\_memoria)

{

// Cargar todos los sectores que corresponden al

// mapa de bits

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);

// Calcular en que sector está el bit modificado

// Escribir el sector en disco

sector=(offset/512);

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

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

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

return(1);

}

# Funciones para la lectura y escritura de los bloques

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

// Lectura y escritura de bloques

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

int writeblock(int block,char \*buffer)

{

int result;

int i;

// Determinar si el sector de boot de la partición está en memoria, si no está en memoria, cargarlo

if(!secboot\_en\_memoria)

{

// Leer el sector lógico 0, donde está

// el sector de boot de la partición

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

secboot\_en\_memoria=1;

}

// Inicio area de datos = setor de inicio de la partición +

// sectores reservados +

// sectores mapa de bits area de nodos i+

// sectores mapa de bits area de datos+

// sectores area nodos i (dr)

inicio\_area\_datos=secboot.sec\_inicpart+secboot.sec\_res++secboot.sec\_mapa\_bits\_nodosi +secboot.sec\_mapa\_bits\_bloques+secboot.sec\_tabla\_nodos\_i;

// Escribir todos los sectores que corresponden al

// bloque

for(i=0;i<secboot.sec\_x\_bloque;i++)

vdwriteseclog(inicio\_area\_datos+(block-1)\*secboot.sec\_x\_bloque+i,buffer+512\*i);

return(1);

}

int readblock(int block,char \*buffer)

{

int result;

int i;

// Determinar si el sector de boot de la partición está en memoria, si no está en memoria, cargarlo

if(!secboot\_en\_memoria)

{

// Leer el sector lógico 0, donde está

// el sector de boot de la partición

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

secboot\_en\_memoria=1;

}

// Inicio area de datos = setor de inicio de la partición +

// sectores reservados +

// sectores mapa de bits area de nodos i+

// sectores mapa de bits area de datos+

// sectores area nodos i (dr)

inicio\_area\_datos=secboot.sec\_inicpart+secboot.sec\_res++secboot.sec\_mapa\_bits\_nodosi +secboot.sec\_mapa\_bits\_bloques+secboot.sec\_tabla\_nodos\_i;

for(i=0;i<secboot.sec\_x\_bloque;i++)

vdreadseclog(inicio\_area\_datos+(block-1)\*secboot.sec\_x\_bloque+i,buffer+512\*i);

return(1);

}

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

// Funciones para el manejo de inodos

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

unsigned int datetoint(struct DATE date)

{

unsigned int val=0;

val=date.year-1970;

val<<=4;

val|=date.month;

val<<=5;

val|=date.day;

val<<=5;

val|=date.hour;

val<<=6;

val|=date.min;

val<<=6;

val|=date.sec;

return(val);

}

int inttodate(struct DATE \*date,unsigned int val)

{

date->sec=val&0x3F;

val>>=6;

date->min=val&0x3F;

val>>=6;

date->hour=val&0x1F;

val>>=5;

date->day=val&0x1F;

val>>=5;

date->month=val&0x0F;

val>>=4;

date->year=(val&0x3F) + 1970;

return(1);

}

// Obtiene la fecha y hora actual del sistema y la

// empaqueta en un entero de 32 bits

unsigned int currdatetimetoint()

{

struct tm \*tm\_ptr;

time\_t the\_time;

struct DATE now;

// Llamada al sistema para obtener la fecha/hora actual

// y guardar el resultado en the\_time

(void) time(&the\_time);

tm\_ptr=gmtime(&the\_time);

// Poner la fecha y hora obtenida en la estructura TIME

now.year=tm\_ptr->tm\_year-70;

now.month=tm\_ptr->tm\_mon+1;

now.day=tm\_ptr->tm\_mday;

now.hour=tm\_ptr->tm\_hour;

now.min=tm\_ptr->tm\_min;

now.sec=tm\_ptr->tm\_sec;

// Convertirlo a un entero de 32 bits y regresar el

// resultado

return(datetoint(now));

}