# Funciones para el manejo de los nodos-i

Cuando creamos un archivo nuevo en el sistema de archivos, va a ser necesario buscar un inodo disponible y esto lo podemos hacer con una función para buscar en el mapa de bits. Una vez encontrado establecer ese inodo con los datos del archivo que queremos crear.

// Escribir en la tabla de nodos-I del directorio raíz, los datos de un archivo

int setninode(int num, char \*filename,unsigned short atribs, int uid, int gid)

{

int i;

int result;

Antes de continuar debe cargarse en memoria el sector de boot de la partición.

Con los datos que están ahí, calcular:

* El sector lógico donde empieza la tabla de nodos-i del directorio raiz.
* También vamos a usar el número de sectores que tiene la tabla de nodos-i

// Si la tabla de nodos-i no está en memoria,

// hay que cargarla a memoria

if(!nodos\_i\_en\_memoria)

{

for(i=0;i<secboot.sec\_tabla\_nodos\_i;i++)

result=vdreadseclog(inicio\_nodos\_i+i,&inode[i\*4]);

nodos\_i\_en\_memoria=1;

}

// Copiar el nombre del archivo en el nodo i

strncpy(inode[num].name,filename,18);

// Asegurando que el último caracter es el terminador (cero)

if(strlen(inode[num].name)>17)

inode[num].name[17]='\0';

// Poner en el nodo I las fechas y horas de creación

// con las fecha y hora actual

inode[num].datetimecreat=currdatetimetoint();

inode[num].datetimemodif=currdatetimetoint();

// Información sobre el dueño, grupo dueño y atributos

// Para propósitos de la práctica, los datos que se

// refieren a dueño, grupo dueño y atributos (permisos)

// no son relevantes.

inode[num].uid=uid;

inode[num].gid=gid;

inode[num].perms=atribs;

// Un archivo nuevo, su tamaño inicial es 0

inode[num].size=0; // Tamaño del archivo en 0

// Establecer los apuntadores a bloques directos en 0

for(i=0;i<10;i++)

inode[num].blocks[i]=0;

// Establecer los apuntadores indirectos en 0

inode[num].indirect=0;

inode[num].indirect2=0;

// Optimizar la escritura escribiendo solo el sector lógico que

// corresponde al inodo que estamos asignando.

// i=num/8;

// result=vdwriteseclog(inicio\_nodos\_i+i,&inode[i\*8]);

for(i=0;i<secboot.sec\_tabla\_nodos\_i;i++)

result=vdwriteseclog(inicio\_nodos\_i+i,&inode[i\*8]);

return(num);

}

// Buscar en la tabla de nodos I, el nodo I que contiene el

// nombre del archivo indicado en el apuntador a la cadena

// Regresa el número de nodo i encontrado

// Si no lo encuentra, regresa -1

int searchinode(char \*filename)

{

int i;

int free;

int result;

Antes de continuar debe cargarse en memoria el sector de boot de la partición.

Con los datos que están ahí, calcular:

* El sector lógico donde empieza la tabla de nodos-i
* También vamos a usar el número de sectores que tiene la tabla de nodos-i

// Si la tabla de nodos i no está en memoria,

// entonces vamos a traer los sectores lógicos de

// los nodos I a memoria

if(!nodos\_i\_en\_memoria)

{

for(i=0;i<secboot.sec\_tabla\_nodos\_i;i++)

result=vdreadseclog(inicio\_nodos\_i+i,&inode[i\*4]);

nodos\_i\_en\_memoria=1;

}

// El nombre del archivo no debe medir más de 18 bytes

if(strlen(filename)>17)

filename[17]='\0';

// Recorrer la tabla de nodos I que ya tengo en memoria

// desde el principio hasta el final buscando el archivo.

i=0;

while(strcmp(inode[i].name,filename) && i<TOTAL\_NODOS\_I)

i++;

if(i>= TOTAL\_NODOS\_I)

return(-1); // No se encuentra el archivo

else

return(i); // La posición donde fue encontrado

}

// Eliminar un nodo I de la tabla de nodos I, y establecerlo

// como disponible

int removeinode(int numinode)

{

int i;

unsigned short temp[512]; // 1024 bytes

// Desasignar los bloques directos en el mapa de bits que

// corresponden al archivo

Antes de continuar debe cargarse en memoria el sector lógico 1 que es el sector de boot de la partición.

Con los datos que están ahí, calcular:

* El sector lógico donde empieza la tabla de nodos-i
* También vamos a usar el número de sectores que tiene la tabla de nodos-i

Asegurar que los sectores de la tabla nodos-I están en memoria, si no están en memoria, cargarlos.

// Recorrer los apuntadores directos del nodo i

for(i=0;i<10;i++)

if(inode[numinode].blocks[i]!=0) // Si es dif de cero

// Si está asignado

{

unassignblock(inode[numinode].blocks[i]);

inode[numinode].blocks[i]=0;

}

// Si el bloque indirecto, ya está asignado

if(inode[numinode].indirect!=0)

{

// Leer el bloque que contiene los apuntadores

// a memoria

readblock(inode[numinode].indirect,(char \*) temp);

// Recorrer todos los apuntadores del bloque para

// desasignarlos

for(i=0;i<512;i++)

if(temp[i]!=0)

unassignblock(temp[i]);

// Desasignar el bloque que contiene los apuntadores

unassignblock(inode[numinode].indirect);

inode[numinode].indirect=0;

}

// Poner en cero el bit que corresponde al inodo en el mapa

// de bits de nodos-i

unassigninode(numinode);

return(1);

}

# Funciones del Sistema de Archivos

int vdopen(char \*filename,unsigned short mode)

{

// Les toca hacerla a ustedes

}

// Esta función se va a usar para crear una función en el

// sistema de archivos

int vdcreat(char \*filename,unsigned short perms)

{

int numinode;

int i;

// Ver si ya existe el archivo

numinode=searchinode(filename);

if(numinode==-1) // Si el archivo no existe

{

// Buscar un inodo en blanco en el mapa de bits (nodos i)

numinode=nextfreeinode(); // Recorrer la tabla

// de nodos i buscando

// un inodo que esté

// libre

if(numinode==-1) // La tabla de nodos-i está llena

{

return(-1); // No hay espacio para más archivos

}

} else // Si el archivo ya existe elimina el inodo

removeinode(numinode);

// Escribir el archivo en el inodo encontrado

// En un inodo de la tabla, escribe los datos del archivo

setninode(numinode,filename,perms,getuid(),getgid());

// assigninode(numinode);

// Poner el archivo en la tabla de archivos abiertos

// Establecer el archivo como abierto

if(!openfiles\_inicializada)

{

// La primera vez que abrimos un archivo, necesitamos

// inicializar la tabla de archivos abiertos

for(i=3;i<16;i++)

{

openfiles[i].inuse=0;

openfiles[i].currbloqueenmemoria=-1;

}

openfiles\_inicializada=1;

}

// Buscar si hay lugar en la tabla de archivos abiertos

// Si no hay lugar, regresa -1

i=3;

while(openfiles[i].inuse && i<16)

i++;

if(i>=16) // Llegamos al final y no hay lugar

return(-1);

openfiles[i].inuse=1; // Poner el archivo en uso

openfiles[i].inode=numinode; // Indicar que inodo es el

// del archivo abierto

openfiles[i].currpos=0; // Y la posición inicial

// del archivo es 0

return(i);

}

// Borrar un archivo del sistema de archivos

int vdunlink(char \*filename)

{

int numinode;

int i;

// Busca el inodo del archivo

numinode=searchinode(filename);

if(numinode==-1)

return(-1); // No existe

removeinode(numinode);

}

// Mover el puntero del archivo a la posición indicada

// a partir de: el inicio si whence es 0, de la posición

// actual si whence es 1, o a partir del final si whence es 2

int vdseek(int fd, int offset, int whence)

{

unsigned short oldblock,newblock;

// Si no está abierto regresa error

if(openfiles[fd].inuse==0)

return(-1); // Regresar ERROR

// La función currpostoptr es una función que a partir de

// la posición actual del apuntador del archivo,

// me regresa la dirección de memoria en el nodo i o en

// en el bloque de apuntadores que contiene el bloque

// donde está el puntero del archivo

oldblock=\*currpostoptr(fd);

if(whence==0) // A partir del inicio

{

// Si el offset está antes del inicio, o después

// del final, regresar un error

if(offset<0||offset>inode[openfiles[fd].inode].size)

return(-1);

**openfiles[fd].currpos=offset;**

} else if(whence==1) // A partir de la posición actual

{

// Validar si no estás quieriendo mover antes del

// inicio del archivo o después del final if(openfiles[fd].currpos+offset>inode[openfiles[fd].inode].size ||

openfiles[fd].currpos+offset<0)

return(-1);

**openfiles[fd].currpos+=offset;**

} else if(whence==2) // A partir del final

{

if(offset>inode[openfiles[fd].inode].size ||

openfiles[fd].currpos-offset<0)

return(-1);

**openfiles[fd].currpos=inode[openfiles[fd].inode].size-offset;**

} else

return(-1);

// Verificamos si la nueva posición del puntero es un

// bloque diferente al que estábamos,si es así hay que

// cargar ese bloque a memoria

newblock=\*currpostoptr(fd);

// Después de mover el puntero, ahora me cambié a otro

// bloque?

if(newblock!=oldblock)

{

// Escribir el bloque viejo

writeblock(oldblock,openfiles[fd].buffer);

// Leer el bloque nuevo

readblock(newblock,openfiles[fd].buffer);

// Indicar el nuevo bloque como bloque actual

// en la tabla de archivos abiertos

openfiles[fd].currbloqueenmemoria=newblock;

}

// Regresamos la posición actual del archivo

return(openfiles[fd].currpos);

}

// Esta es la función más difícil,

int vdwrite(int fd, char \*buffer, int bytes)

{

int currblock;

int currinode;

int cont=0;

int sector;

int i;

int result;

unsigned short \*currptr;

// Si no está abierto, regresa error

if(openfiles[fd].inuse==0)

return(-1);

currinode=openfiles[fd].inode;

// Copiar byte por byte del buffer que recibo como

// argumento al buffer del archivo

while(cont<bytes)

{

// Obtener la dirección de donde está el bloque que corresponde

// a la posición actual

currptr=currpostoptr(fd);

if(currptr==NULL)

return(-1);

// Cuál es el bloque en el que escibiríamos

currblock=\*currptr;

// Si el bloque está en blanco, dale uno

if(**currblock==0**)

{

currblock=nextfreeblock();

// El bloque encontrado ponerlo en donde

// apunta el apuntador al bloque actual

\*currptr=currblock;

assignblock(currblock); // Asignarlo en el mapa de bits

// Escribir el sector de la tabla de nodos i

// En el disco

sector=(currinode/8);

result=vdwriteseclog(inicio\_nodos\_i+sector,&inode[sector\*8]);

}

// Si el bloque de la posición actual no está en memoria

// Lee el bloque al buffer del archivo

if(openfiles[fd].currbloqueenmemoria!=currblock)

{

// Leer el bloque actual hacia el buffer que

// está en la tabla de archivos abiertos

readblock(currblock,**openfiles[fd].buffer**);

// Actualizar en la tabla de archivps abiertos

// el bloque actual

openfiles[fd].currbloqueenmemoria=currblock;

}

// Copia el caracter al buffer

openfiles[fd].buffer[openfiles[fd].currpos%**TAMBLOQUE**]=buffer[cont];

// Incrementa posición actual del actual

openfiles[fd].currpos++;

// Si la posición es mayor que el tamaño, modifica el tamaño

if(openfiles[fd].currpos>inode[currinode].size)

inode[openfiles[fd].inode].size=openfiles[fd].currpos;

// Incrementa el contador

cont++;

// Si se llena el buffer, escríbelo

if(openfiles[fd].currpos%**TAMBLOQUE**==0)

writeblock(currblock,openfiles[fd].buffer);

}

return(cont);

}

int vdread(int fd, char \*buffer, int bytes)

{

// Ustedes la hacen

}

int vdclose(int fd)

{

// Ustedes la hacen

}