Laboratorio 10 - Sistema de Archivos

xv6 organiza el disco de la siguiente manera (ver archivo `fs.h` y `mkfs.c`):

`[ boot block | super block | log | inode blocks | free bit map | data blocks ]`

El primer bloque, `boot block`, es el sector de arranque del sistema. El segundo bloque, `super block`, contiene información acerca del sistema de archivos. Luego hay un conjunto de bloques que utiliza el sistema de sistema de *logging* de *xv6*. A continuación de estos, se encuentra otro conjunto de bloques, destinados a almacenar los i-nodos. Seguido a estos, estan los bloques que almacenan el *bitmap* para administrar el espacio libre en disco.

Al iniciar la ejecución, *xv6* presenta una línea con información acerca de la organización del disco, que indica, entre otras cosas, el número de total de bloques y el número de bloque donde empieza cada uno de los conjuntos anteriores:

```
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
```

Ejercicio 1: Buffering

El programa `write_bytes.c` escribe la cantidad de bytes indicada en un archivo. Por ejemplo, el siguiente comando escribe 100 Mb en el archivo `tmp.txt` usando un buffer de 4096 bytes:

```
$ ./write_bytes tmp.txt 104857600 4096
```

1. Obtener un promedio de tiempo de ejecución del programa, usando el comando `time`, al crear un archivo de 100 Mb, con tamaños de buffer de 256, 1024, 4096 y 8192 bytes. Presentar una tabla que indique el tiempo total, de usuario, de sistema y uso de la CPU. Realizar 10 ejecuciones para obtener el promedio. Explicar los resultados.

Ejercicio 2: Enlaces simbólicos

¿Cómo podría implementarse enlaces simbólicos en xv6? Describir brevemente como sería la implementación, indicando que modificaciones y agregados haría en el sistema.

Ejercicio 3: E/S de disco en xv6

Agregar en las funciones `bwrite()` y `bread()`, en el archivo `bio.c`, la siguiente invocación a `cprintf()`. De esta manera se muestra un mensaje cada vez que se escribe o lee un bloque en disco:

```
cprintf("bwrite block%d\n", b->blockno); // usar "bread" en la funcion bread()
```

Luego, compilar y ejecutar *xv6*, y ejecutar el comando `echo > a`. Comprobar que aparezca por pantalla los bloques que se escriben o leen.

Modificar la invocación a `cprintf()` para indique también a donde pertenece el bloque que está siendo leído o modificado en disco. Para averiguar como esta organizado el disco consultar los archivos `fs.h` y `mkfs.c`. Por ejemplo:

```
$ echo > a
bwrite block 3 (log)
...
$
```

1. Ejecutar el comando `echo "hola" > a.txt`. Explicar las lecturas y escrituras que se realizan.

2. Comentar el `cprintf()` de la función `bread()`. Ejecutar `make clean && make qemu-nox` para volver a generar la imagen del disco. A continuación, iniciar nuevamente xv6 y ejecutar los siguientes comandos. Para cada uno indicar cuáeles son los bloques que se modifican y por qué razón:

```
$ mkdir d
$ echo > d\a
$ echo a > d\a
$ rm d\a
$ rm d
```

Ejercicio 4: Incrementar el tamaño máximo de un archivo en *xv6*

Un *i-nodo* en *xv6* contiene 12 bloques de acceso directo, más un bloque de indirección sencilla que agrega 128 bloques adicionales (512 / 4), por lo que el máximo número de bloques en disco que puede ocupar un archivo es 140 (12 + 128). Como el tamaño de un bloque es igual al de un sector (`BSIZE = 512`), un archivo en *xv6* puede ocupar como máximo 140 sectores en el disco (71680 bytes).

En este ejercicio se incrementará el tamaño máximo de un archivo en *xv6*, agregando soporte en la estructura de *i-nodo* para un bloque de indirección doble.

Preliminares

1. En el archivo `Makefile` de *xv6* indicar que simule un solo CPU (`CPU := 1`), y agregar la opción `-snapshot` en la definición de `QEMUOPTS`. Estos cambios mejoran la performance de *xv6* al generar archivos grandes, y utilizar solo una CPU facilita la evaluación.
2. Modificar `FSSIZE` en el archivo `param.h` para que sea igual a 262144 sectores. Esto incrementa el tamaño de la imagen de disco a 128 Mb (262144 * 512 bytes).
3. Copiar el archivo `big.c` en el directorio de *xv6*, y agregarlo a la lista `UPROGS` en el `Makefile`. Este programa al ejecutarse crea un nuevo archivo, con un tamaño tal que ocupe un número determinado de sectores en el disco.
4. Compilar y ejecutar *xv6*. Luego, ejecutar el comando `big` con 200 sectores como parámetro. Debe retornar que sólo 140 sectores fueron escritos, ya que es el máximo tamaño posible del archivo.

Qué tener en cuenta

El formato de un *i-nodo* en disco es establecido por la estructura `struct dinode`, definida en el archivo `fs.h`. Prestar atención a `NDIRECT`, `NINDIRECT`, `MAXFILE` y el arreglo `addrs[]`.

La función `bmap()`, en el archivo `fs.c`, permite recuperar los datos de una archivo en el disco. Esta función es invocada tanto en la lectura como la escritura de un archivo. Para este último caso, `bmap()` reserva nuevos bloques según sea necesario.

Notar que `bmap()` maneja dos tipos de números de bloques. El argumento `bn` indica un número lógico de bloque, que es relativo al inicio del archivo. Sin embargo, los números de sectores almacenados en el arreglo `addrs[]` del *i-nodo* corresponden con números de sectores en el disco, que pueden no ser consecutivos.

Modificaciones a realizar

Modificar `bmap()` para que implemente el bloque de indirección doble, además del bloque de indirección sencilla y los bloques directos.

No se debe modificar el tamaño del *i-nodo*, si no que, en cambio, se debe alterar para que tenga 11 bloques directos (en lugar de 12). De esta manera, el elemento 10 del arreglo `addrs[]` será el bloque indirecto sencillo, y el último elemento del arreglo será la dirección del nuevo bloque de indirección doble.

Se debe modificar también, en el archivo `mkfs.c`, la función `iappend()` de manera similar. Este programa genera la imagen de disco inicial (archivo `fs.img`) para la máquina virtual, y crea los *i-nodos* de los programas en disco.

Tips:

- Liberar cada bloque luego de utilizarlo, utilizando la función `brelse()`.
- Sólo se deben reservar nuevos sectores en disco a medida que sean necesarios.
- Si el sistema de archivos se corrompe, eliminar el archivo `fs.img`.

Entrega

Agregar al repositorio del Laboratorio una copia del archivo `fs.c` modificado.

¡Fín del Laboratorio 10!