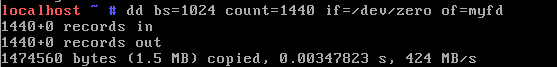
|  |  |
| --- | --- |
| **Block name** | **Block location** |
| Super block | 0000400 |
| Group descriptor | 0000800 |
| IBM | 0002400 |
| DBM | 0002000 |
| Inode table | 0002800 |

1) Make a virtual floppy disk

# dd bs=1024 count=1440 if=/dev/zero of=myfd

will make a virtual floppy disk of size 1.44MB with name “myfd”.

“dd” is a command to write data into disk. “bs” is block-size. bs=1024 means 1 block is 1024 byte. “count” is the number of blocks to write. count=1440 means write 1440 blocks, which is 1440\*1024=1440KB=1.44 MB. “if” is input file to read data from. /dev/zero is a special file that gives out zeros when being read. “of” is the output file.

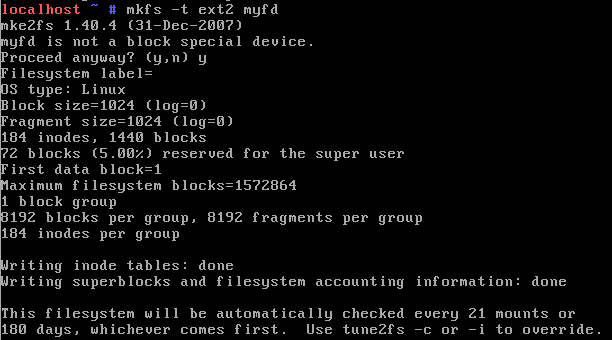


dd명령어를 써서 총 1440개의 1024bytes 블록을 만들어줬다.

2) Format

# mkfs -t ext2 myfd

will format myfd with ext2 file system.



mkfs 명령어를 사용해서 myfd에 ext2 파일 시스템을 만들어줬다.

3) Mount

# mkdir temp

We need an empty directory to mount myfd.

temp 디렉토리는 이미 있기 때문에 따로 만들어주지 않았다.

# mount -o loop myfd temp

will mount myfd to temp directory. Since myfd is not a physical disk but a file that contains a disk image, we need to use –o loop option.



디스크를 맘대로 조작하기 위해서 mount 명령어를 사용해 temp에 myfd를 연결시켜줬다.

4) Make some files

# cd temp

# echo korea > f1

will make f1 in temp.



echo 명령어를 사용해서 f1을 temp안에 만들어줬다.

5) Read

# cd ..

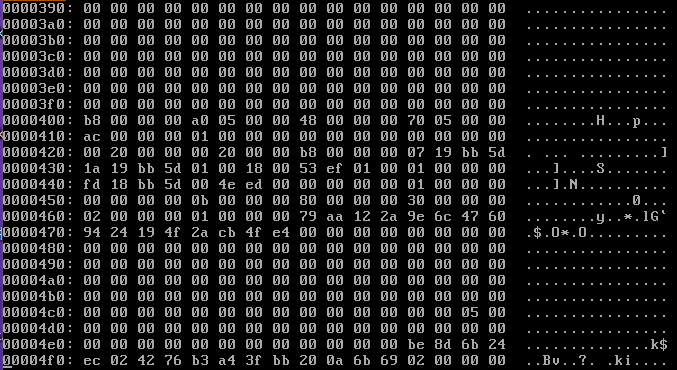
# umount temp

We need “umount temp” to write the change in the virtual disk, myfd.

# xxd -g1 myfd > x

“-g1” option will display each byte of myfd separately.

# vi x



xxd 명령어를 사용한 뒤 x로 들어가본 결과, 모든게 다 비트 단위로 끊어져서 x에 들어가있었다.

6) Read superblock. Superblock starts at offset 1024(400h). Find the superblock and confirm the magic number(0xEF53), block size, and the first block number

*typedef struct // super block*

*{*

*u16 m\_magic; // 38-39*

*u32 m\_log\_block\_size; // 18-1B. block size=1024\*pow(2, m\_log\_block\_size)*

*u32 m\_first\_data\_block; // 14-17*

*…*

*} SuperBlock;*



super block은 400h에 있어서 x에서 0000400을 찾았다.

magic number : 0xEF53 (438)

first block number : 0x00000001 = 1

block size : m=0x0=0 (00 00 00 00) (418), 1024\*2^m = 1024bytes. (block size = 1024 \* = 1024 Byte)

7) Read the group descriptor and confirm the block location of IBM, DBM, and inode table.



블럭 하나가 400h라 블럭넘버\*400하면 무조건 그 블럭의 위치가 나온다. group descriptor은 2번 블록을 사용하고 있기 때문에 2\*400h인 800h에 가보면 볼 수 있다.

DBM : 8\*400h = 2000h

IBM : 9\*400h = 2400h

inode table : a\*400h = 2800h

8) Read the IBM and DBM, and confirm the inode numbers and block numbers in use. Draw the layout of myfd disk that shows the block location of all meta blocks: super block, group descriptor, IBM, DBM, and inode table.

DBM



ff ff ff ff ff 7f를 2진수로 바꿔줬을 때, 1의 개수가 사용중인 블럭의 개수가 된다. 만약에 f1이 없다면 7f가 아닌 3f가 돼서 사용중인 블럭 개수는 46개였겠지만, f1이 47번 블록을 사용하게 돼서 사용중인 블럭 개수는 47개가 된다.

IBM



IBM은 inode bit map으로 DBM과 비슷하게 분석해봤을 때 어느 노드들이 사용중인지 알 수 있다. f1을 만들기 전에는 ff 07이여서 총 11개의 노드가 사용중이였지만, f1이 만들어진 이후로 f1이 inode 12번을 사용하게돼서 사용중인 노드가 12개로 늘었다.

inode table



inode table은 2800h부터 시작하고, inode 2의 블록 넘버인 21도 볼 수 있었다. (21 is in 2880 + 28)

(inode 하나당 80h씩 씀. 블록넘버 = 28부터 쭉. 강의노트 typedef struct 참고)

*typedef struct // inode*

*{*

*u16 m\_mode; // 0-1*

*u16 m\_uid; // 2-3*

*u32 m\_size; // 4-7*

*u32 m\_atime; // 8-B*

*u32 m\_ctime; // C-F*

*u32 m\_mtime; // 10-13*

*u32 m\_dtime; // 14-17*

*u16 m\_gid; // 18-19*

*u16 m\_links\_count; // 1A-1B*

*u32 m\_blocks; // 1C-1F. shows num of data blocks for this file in units of 512 bytes*

*u32 m\_flags; // 20-23*

*u32 m\_reserved1; // 24-27*

*u32 m\_block[15]; // block location of this file*

*u32 m\_generation;*

*u32 m\_file\_acl;*

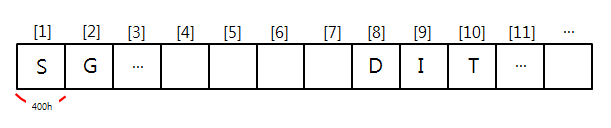
*u32 m\_dir\_acl;*

*u32 m\_faddr;*

*u32 m\_reserved2[3];*

*} Inode;*

|  |  |
| --- | --- |
| **Block name** | **Block location** |
| Super block | 0000400 |
| Group descriptor | 0000800 |
| IBM | 0002400 |
| DBM | 0002000 |
| Inode table | 0002800 |



* 1번 S : Super Block
* 2번 G : Group Descriptor
* 8번 D : DBM
* 9번 I : IBM
* 10번 T : Inode Table