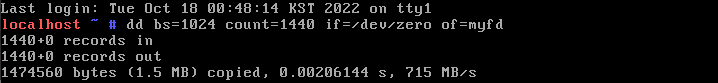
**오퍼레이팅시스템 9주차 과제**

**12180626 성시열**

6. (homework) Read the disk and analyze the contents of the meta blocks.

1) Make a virtual floppy disk

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



주어진 명령어를 통해 virtual floppy disk를 생성한다.

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 (this "block" is not the file system block, it is just the unit of data transfer in "dd" command). “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. Note bs is NOT file system block size. You can make same disk with bs=1 as below:

# dd bs=1 count=1474560 if=/dev/zero of=myfd

bs=1일 때는 count를 다음과 같이 늘려준다.

1-1) Check the content of myfd. What data do you have in it? Check address 0x400.

# xxd myfd > x

# vi x

/400

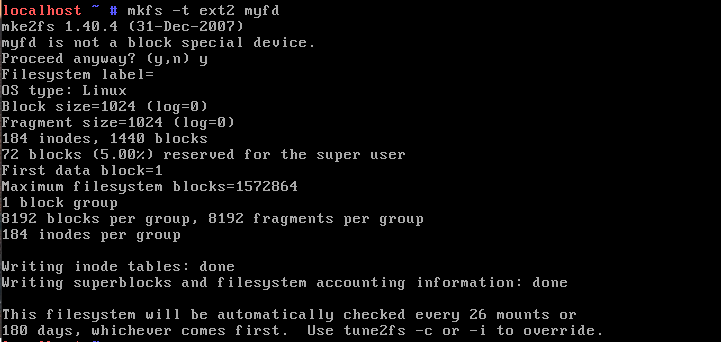


위 명령어를 통해 실행한 뒤 super block의 시작 주소인 400으로 이동하여 확인하였으나 아직은 모두 0인 것을 확인하였다.

2) Format

# mkfs -t ext2 myfd

will format myfd with ext2 file system.

’

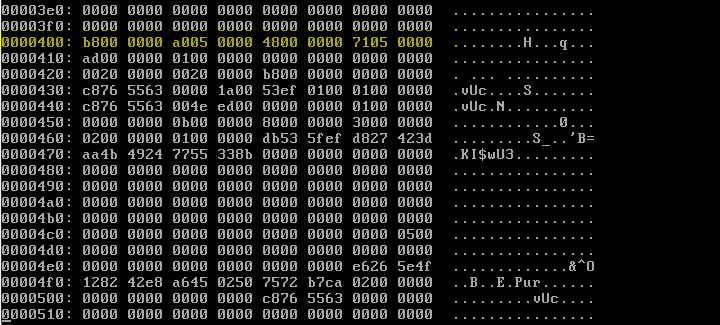
주어진 명령어를 통해 myfd 이름의 ext2 버전 fs을 생성한다.

2-1) Now what data do you have in myfd? First 4 byte at address 0x400 must be b8 00 00 00. What is the meaning of them? Who has written that data in myfd?

# xxd myfd > x

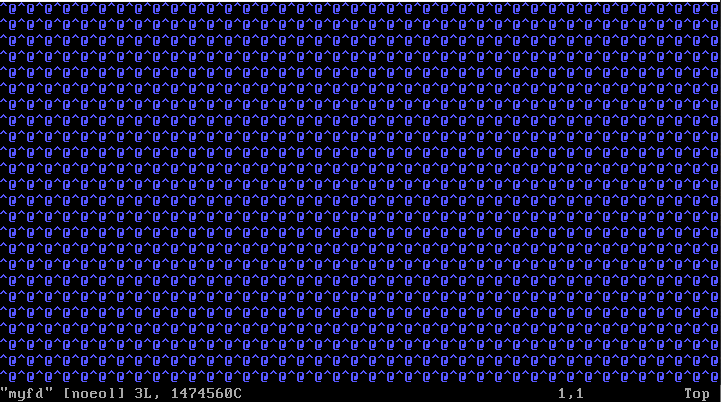
# vi x

/400



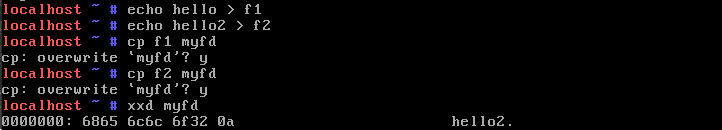
다시 한번 super block의 시작 주소인 400으로 이동하여 확인해본 결과 표시한 부분에 값이 변경됨을 확인하였다. “b800 0000”은 리틀 엔디안 방식이므로 00 00 00 b8, 10진수로 하면 184이다. 이는 최대 184개의 inode를 사용할 수 있다는 뜻이다.

2-2) myfd is actually a simple file with EXT2 file system written in it. However, "vi myfd" will not show the content. Why is that? How can you see the content of myfd?



문제에서 제시된대로 vi myfd를 실행해보았으나 위와 같이 해석할 수 없는 문자들이 출력되었다. myfd는 16진수로 이루어진 파일이므로 내용을 확인하려면 xxd myfd > x를 통해 x에 파일을 “16진수”로 저장한 뒤 이를 확인해야 한다.

2-3) myfd is a virtual disk which means you can store a file in it as if it is a regular disk. You may think following will store two files (f1 and f2) in myfd disk. Explain why it is not working.



주어진 명령어를 실행시켰다. Myfd가 생성되어있던 중 f1과 f2를 입력해준 것이므로 오버라이트를 진행하였다.

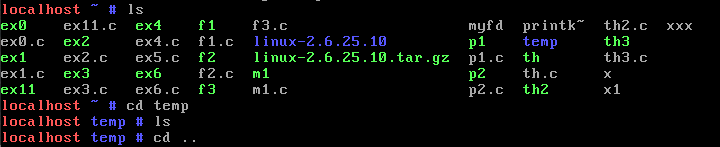
16진수로 myfd 내용을 확인해본 결과 hello라는 텍스트가 저장된 파일이 myfd에 저장되고 두번째 복사했던 hello2가 모두 저장될 것이라 예상할 수 있지만, hello가 들어가있던 myfd 파일에 hello2를 overwrite하여 저장하였으므로 가장 나중에 복사를 진행한 파일만이 저장되어있다. myfd를 파괴했으므로 1,2 과정을 진행하여 myfd를 원상복귀한 후 진행한다.

3) Mount

To store a file in myfd, you have to connect the file tree in myfd to the system file tree via mounting. Regenerate myfd and do followings.

# mkdir temp

We need an empty directory to mount myfd. If temp is already there, skip this step.



빈 directory인 temp가 생성되어 있음을 확인하였다.

# mount -o loop myfd temp

will connect myfd to temp directory, which is called mounting. Since myfd is not a physical disk but a file that contains a disk image, we need to use –o loop option. This command will connect myfd disk to /temp directory and you can access "myfd" disk via /temp.



mount를 통해 myfd 를 temp에 연결한다.

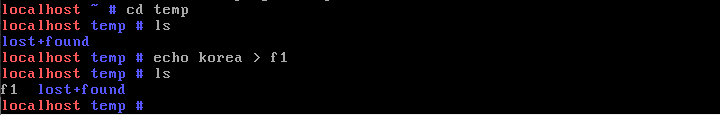
4) Make some files in myfd

# cd temp --- go to myfd

# ls --- check myfd is empty

# echo korea > f1 --- make f1 in it

# ls --- check f1



명령어 실행 결과 temp directory에 lost+found라는 directory가 생성됨을 확인하였고 해당 위치에 f1 파일을 생성. 내용은 korea로 저장하였다.

5) Analyze myfd

# cd ..

# umount temp

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



상위 디렉토리로 이동 후 umount를 통해 temp와의 연결을 끊어준다.

# xxd -g1 myfd > x

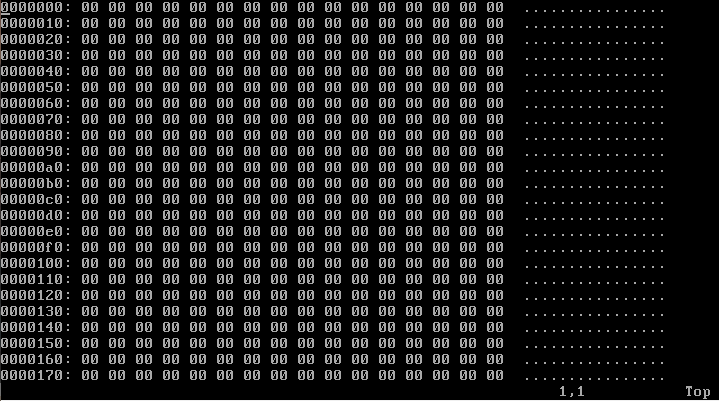
“-g1” option will display each byte of myfd separately.

# vi x

(if it doesn't start at line 0, type ":1" to go to line 1)



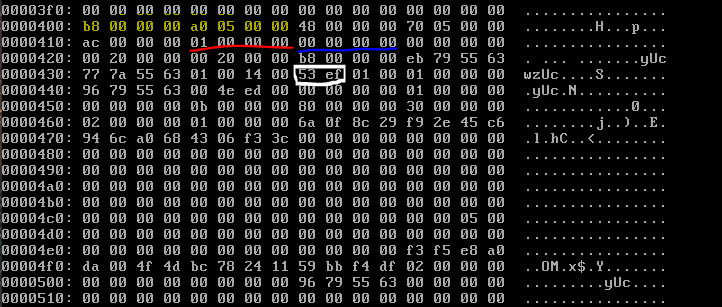
위에서 생성한 파일을 디스크에서 사용하기 위해 umount를 진행하였다. myfd에 저장된 내용을 -g1 옵션을 통해 myfd의 각 byte를 개별적으로 보여준다.



출력 결과 위와 같이 주소에 맞는 16진수들이 출력된 것을 확인할 수 있다.

6) Read superblock. Superblock starts at offset 1024(400h) always. Find the superblock and read the total number of inodes in this disk (m\_inodes\_count), total number of blocks in this disk (m\_blocks\_count), the magic number (m\_magic), block size (1024\*), and the first data block number (m\_first\_data\_block). Every multi-byte number is stored in little endian, and you have to reverse the byte order to compute the actual value.

super block의 시작 주소인 400으로 이동하여 코드를 확인한다.



표시된 순서대로 해석해보자

|  |  |
| --- | --- |
| *u32 m\_inodes\_count //0-3* | b8 00 00 00 => 00 00 00 b8 => 184  최대 184개 파일의 inode를 저장할 수 있다. |
| *u32 m\_blocks\_count //4-7* | a0 05 00 00 => 00 00 05 a0 => 1440  최대 1440개 block을 저장할 수 있다. |
| *u16 m\_magic //38-39* | 53 ef => EF 53  magic number는 EF 53이다. |
| *u32 m\_log\_block size //18-1B* | 00 00 00 00 => 0  Block size = 1024 \* 20 = 1024byte이다. |
| *m\_first\_data\_block //14-17* | 01 00 00 00 => 00 00 00 01 => 1  first block number는 1이다. |

7) Read the group descriptor and find the block number of DBM, IBM, and inode table. The group descriptor is located in the next block after the superblock. If the superblock is at block 0, the group descriptor will start at block 1, if the superblock is at block 1, the group descriptor start at block 2, etc. The address of block x will be x\*block\_size.

Super block의 block size가 1024byte라는 것을 파악하였다. 이는 16진수로 하면 0x400이다. super block의 시작 주소였던 0x400에서 0x400(1024byte)을 더한 0x800으로 이동한다.



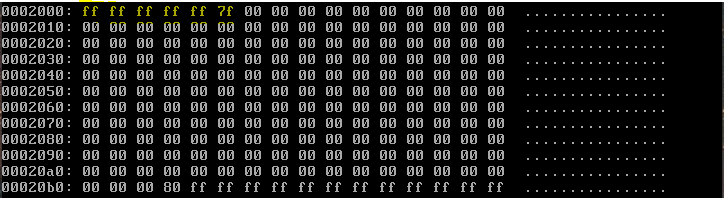
u32는 4byte씩 코드를 확인한다.

|  |  |
| --- | --- |
| *u32 m\_block\_bitmap*  *// block location of DBM* | 08 00 00 00 => 00 00 00 08 => 8 |
| *u32 m\_inode\_bitmap*  *// block location of IBM* | 09 00 00 00 => 00 00 00 09 => 9 |
| *u32 m\_inode\_table*  *// block location of inode table* | 0a 00 00 00 => 00 00 00 0a => 10 |

8) Read the DBM, IBM, and find 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(Data block Bit Map)의 위치가 8임을 확인하였고 이는 byte size가 0x400이므로

8 \* 0x400 =0x2000이다. 해당 위치로 이동한다.



해당 위치에서 ff ff ff ff ff 7f를 발견할 수 있었다. 이를 2진수로 나타내면

11111111 11111111 11111111 11111111 11111111 01111111이다.

값은 각 byte의 오른쪽부터 해석한다.

6번째 7f가 가리키는 값은 block41 ~ block48중 block48이 비어있는 것을 의미한다. 즉 block 41번 ~ block 47번까지 1이라는 것을 의미한다.

IBM(Inode Bit Map)의 위치가 9임을 확인하였고 이는 byte size가 0x400이므로

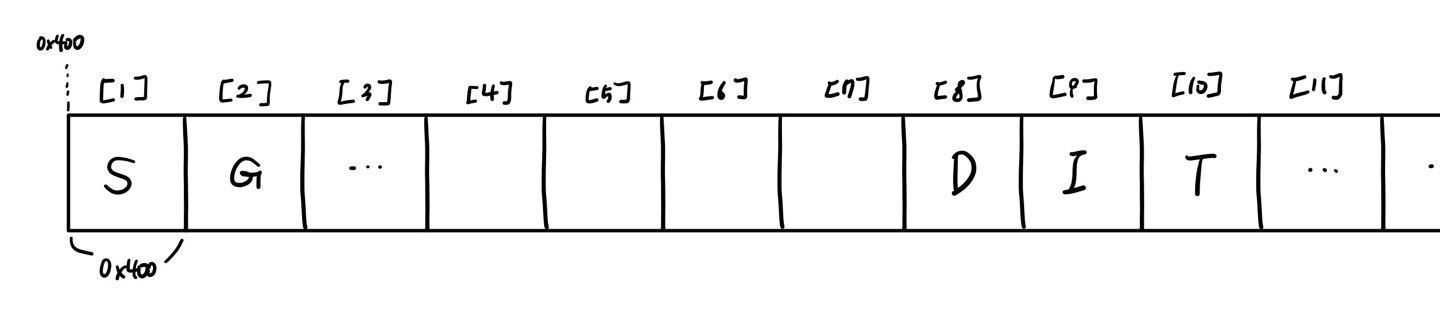
9 \* 0x400 =0x2400이다. 해당 위치로 이동한다.



해당 위치에서 ff 0f를 발견할 수 있었다. 이를 2진수로 나타내면

11111111 00001111 이다. block 1~12번이 사용 중이라는 것을 알 수 있다.

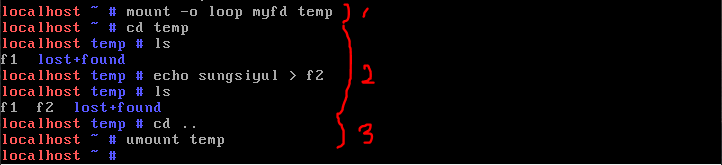
(8 7 6 5 4 3 2 1) (16 15 14 13 12 11 10 9) 순서라고 이해하면 됨.



S : Super Block G : Group Descriptor D : DBM

I : IBM T : Inode Table 이다.

8-1) Create another file f2 in myfd with vi. What changes can you observe in DBM and IBM? Guess what is the inode number for f2 and what is the block number assigned to it.



F2 파일을 생성하기 위해 다시 myfd와 temp를 연결하고 temp directory에 sungsiyul이라는 텍스트가 포함된 f2파일을 생성하였다. 이후 연결을 끊어주었다.



해당 내용을 x에 저장한 뒤 x를 확인하였다.



m\_first\_data\_block 을 확인해본 결과 이전과 같음을 알 수 있다.



DBM IBM을 확인하였다. 이전과 같음을 확인하였다.

DBM(Data block Bit Map)의 위치가 8임을 확인하였고 이는 byte size가 0x400이므로

8 \* 0x400 =0x2000이다. 해당 위치로 이동한다.



해당 위치에서 ff ff ff ff ff ff를 발견할 수 있었다. 이를 2진수로 나타내면

11111111 11111111 11111111 11111111 11111111 11111111이다.

비어있는 block이 없음을 의미한다. 이는 f2가 추가로 생성되면서 block을 차지했고, 이를 통해 빈 block이었던 “block 48”에 f2가 존재함을 알 수 있다.

IBM(Inode Bit Map)의 위치가 9임을 확인하였고 이는 byte size가 0x400이므로

9 \* 0x400 =0x2400이다. 해당 위치로 이동한다.



해당 위치에서 ff 1f를 발견할 수 있었다. 이를 2진수로 나타내면

11111111 00011111 이다. block 1~13번이 사용 중이라는 것을 알 수 있다.

Inode 번호는 파일 하나당 한 개가 할당되므로 “inode 13”에 f2가 할당됨을 추측할 수 있다.