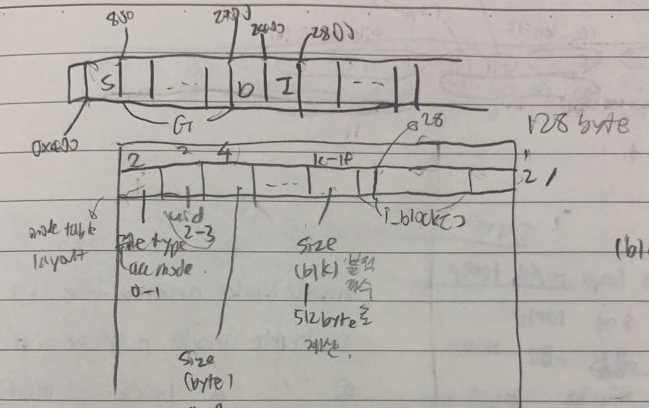


root inode (2880h) -> block number 21번 발견 -> 21\*400h = **8400h = root directory file (모든 파일의 inode 번호와 정보들적혀있음) -> inode 번호 찾기 -> 2800 + 80(inode – 1) -> inode table에서 블록번호 (+28h) 찾기 -> 블록번호 \* 400h = file.**

9) Read the inode table and find the block location of the root directory file. What is the byte size and block size of this file? Who is the owner of this file?



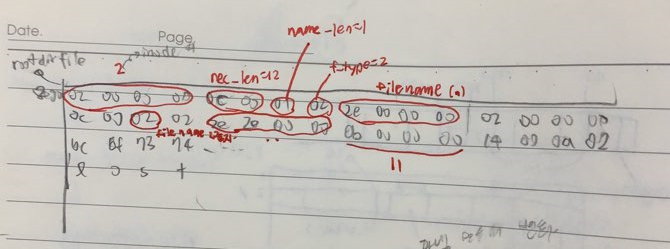


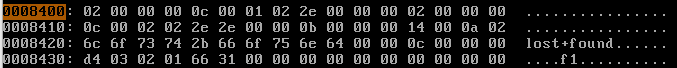
(inode table 시작 : a\*400h = 2800h, root inode table은 2880h에 존재. 28번째에 block number 있음)

root directory file의 블록 번호는 21이고, 실제 위치는 21\*400h인 8400에 있다. 이 파일의 바이트 사이즈는 0x400으로 1024 (decimal로) 바이트가 된다. 블록 사이즈는 2라고 적혀있지만 이것은 512바이트 기준으로 2이고, 실제로 myfd는 한 블록당 1024바이트를 쓰기 때문에 실제 블록 사이즈는 1이다. 이 파일의 owner는 uid와 같기 때문에 0이 된다.

10) Read the root directory file. How many member files it has? What are the inode numbers and file names of them? Go to the inode table and find the block location of each member file.

root directory file is in 8400h. (21\*400)







root directory file은 자기자신인 ., 부모인 .., 시스템이 만든 lost+found와 내가 만든 f1으로 총 4개가 있다. 각각의 inode number은 2, 2, 11과 12이다.

inode table

. & ..



lost+found



f1



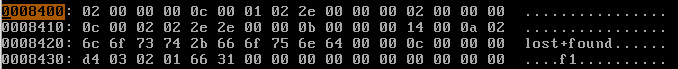
블록번호는 각 파일의 inode table의 시작에서 28을 더한 위치에 있다. .과 ..은 33, lost+found는 34, 그리고 f1은 47번에 있는걸 확인할 수 있다. (decimal로 바꿨을 때.)

HEX : 21, 22, 2f

11) Read the member file and confirm the contents.

위에서 찾은 블록번호에 곱하기 400h를 해줘서 각각의 member file을 읽었다. (ex : 2fh \* 400h = 0xbc00)

. &..



lost+found



lost+found 파일 안에는 자기 자신인 .과 부모인 ..이 들어있는걸 확인할 수 있었다. 이때 .의 inode number은 자기자신인 lost+found의 inode number와 같다는 것도 확인할 수 있었다.

f1



f1도 블록번호 \* 400h를 해서 갔을 때 f1의 내용인 korea가 그대로 적혀있는걸 볼 수 있었다.

12) You can see all files including hidden ones with "ls -a". Confirm you can see all files you found in the file system with this command.

# ls -a



# mount -o loop myfd temp

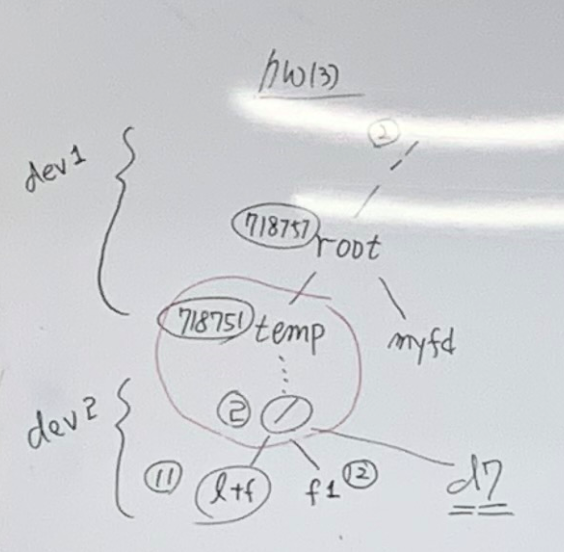
myfd를 temp에 연결시켜주고 temp로 들어가서 ls -a를 해준 결과, 내가 파일 시스템에서 찾았던 모든 파일을 볼 수 있었다.

13) You can see the inode number of a file with "ls -i". Confirm the inode numbers of all files.

# ls -ai



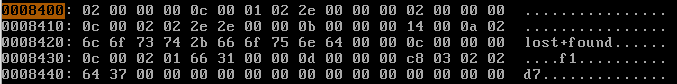
ls -ai를 해서 파일의 inode 번호를 확인해보았다. 여기서 ..만 inode 번호가 2가 아닌 713857이 나오는데, 이것은 원래 temp 밑에 루트가 있고 그 밑에 파일들이 있지만 사용자에게는 temp 밑에 바로 파일들이 있는 것으로 보여져서 부모의 inode 번호를 찍으라고 하면 2가 아닌 temp의 부모인 root의 inode 번호를 찍게 돼서 ..의 inode 번호만 713857이 나오게 돼서 그렇다.



14) Make another file in your virtual disk. Confirm the changes in the file system: IBM, DBM, Inode table, and root directory file. Now delete this file (with "rm" command). What happens to the file system? How can you recover this file?

14-1) Make a new directory (d7) in the root directory with "mkdir" command. Show the disk block content of the root directory file and find out the inode number of d7.

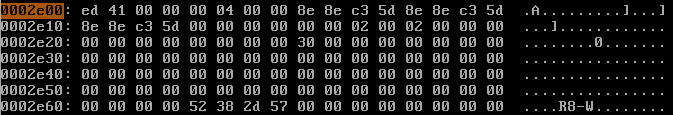




//inode number of d7 is d

mkdir로 d7 디렉터리를 만들고 root directory file로 가서 d7의 inode 번호를 확인해본 결과, d7의 inode 번호는 13이라는걸 알 수 있었다. (정 시간 없으면 ls -ai로 아이노드 번호 알아보기)

14-2) Show the inode content of d7. What is the block location of d7? Show the block content of d7. What files do you have in d7?



d7의 블록위치는 30이라는걸 알 수 있다.

(30h \* 400h = 0xc000)



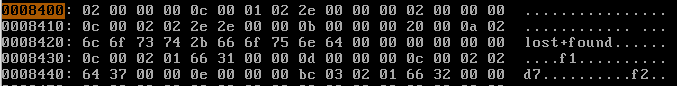
d7의 block content를 봐보면 d7 자기 자신인 . 파일과 d7의 부모인 ..파일이 존재하는걸 알 수 있다.

14-3) Run "mv f1 d7/f2" and show the changes in the root directory file, d7 file, and inode table.

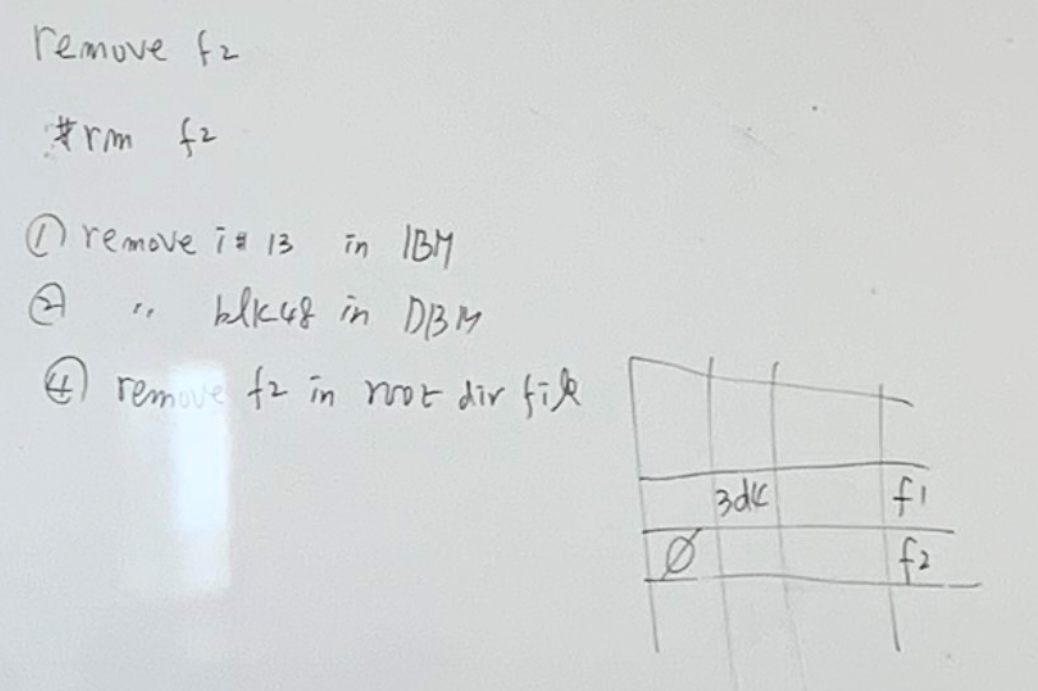


echo를 사용해 hello를 f2에 집어넣어줬고, 그 다음 명령으로 f1을 지우고 f2를 d7에 넣어줬다.

root directory file



f1의 inode 번호가 0으로 세팅됐기 때문에 f1은 지워졌다고 할 수 있다.



d7 file



d7에는 기존에 있던 자기 자신인 .파일과 부모인 ..파일 있고, 거기에 새롭게 f2가 추가된걸 확인할 수 있었다.

inode table

. & ..



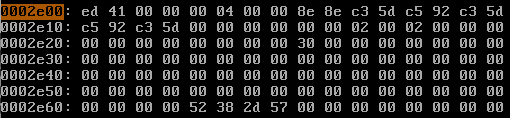
lost+found



f1



d7



f2



inode 테이블을 보면 다른 정보는 조금씩 바꼈지만 블록번호의 정보는 바뀌지 않고 그대로 있는걸 확인할 수 있었다. 그리고 실제로 f1의 블록위치에 가본 결과, korea가 그대로 f1에 들어있는걸 확인할 수 있었다.

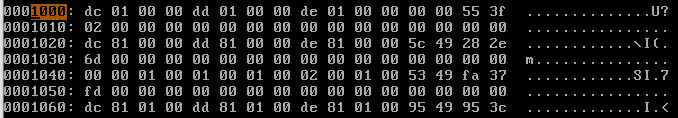
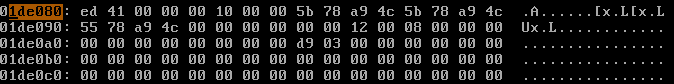
15) Examine the file system in the hard disk (/dev/sda3) and find file names in the root directory

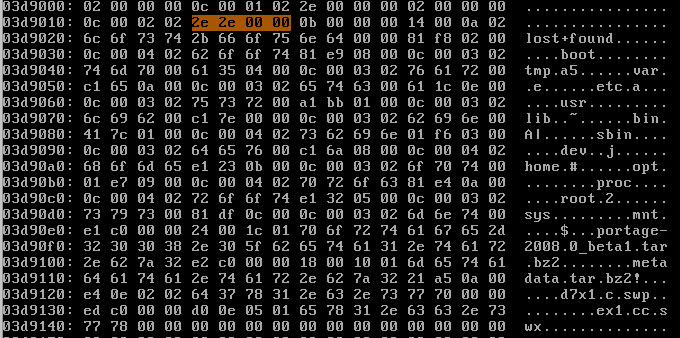
# dd bs=1024 count=8000 if=/dev/sda3 of=myhd

# xxd –g1 myhd > x

# vi x

  
블락 사이즈를 알기 위해서 24-27바이트를 확인해보았다. 2이므로 앞선 과제에서와는 다르게 블락 사이즈가 1024\*2^2 = 4096(1000h) 임을 알 수 있다.  
그리고 첫번째 블락의 시작 위치는 노란 박스 인데 0이다. 이 의미는 슈퍼 블락이 0부터 시작된 것을 알 수 있다. 따라서 group descriptor의 위치는 슈퍼블락이 끝나는 위치 1000h인 것을 알 수 있다. (왜냐면 1\*1000 = 1000h)

(group descriptor)  
  
여기서로부터 DBM, IBM, inode table의 위치를 알 수 있다.  
DBM = 01dc \* 1000h = 1DC000  
IBM = 01dd \* 1000h = 1DD000  
inode table = 01de \* 1000h = 1DE000  
inode table을 확인하기 위해 1DE000으로 이동해본다.  
이 중 root는 2번째 inode이므로 1DE080에 위치한다.  
  
root directory file의 위치는 3d9에 위치함을 알 수 있다.  
3d9 \* 1000h = 3D9000



부모의 이름인 2e 2e 00 00을 검색해서 root directory file을 찾아냈다. 그 안에서 찾은 파일 이름으로는 ., .., lost\_found, boot, tmp, var, etc, usr, lib, ~, bin, sbin, dev, home, opt, proc, root, sys, mnt, $, portage-2008.0\_beta1, metadata, d7x1, ex1이 있었다.