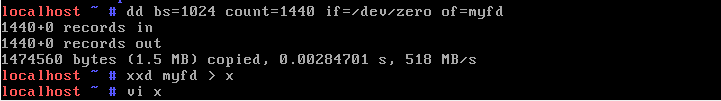
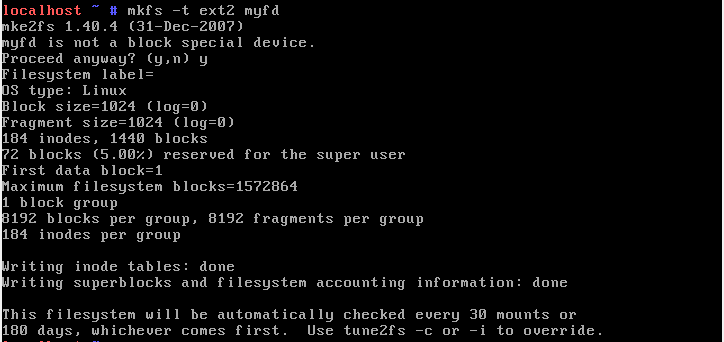
**오퍼레이팅시스템 10주차**

**12180626 성시열**

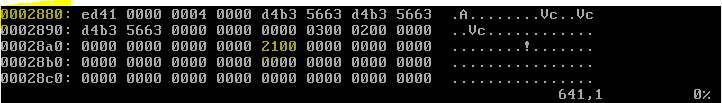
9) Read the inode table and find the block location of the root directory file. What is the byte size and block size of this file? Who is the owner of this file?



virtual floppy disk를 생성한다.



myfd 이름의 ext2버전 fs를 생성한다.



Inode 테이블로 이동하여 root directory file로 이동하여 확인하였다.

표시된 위치에 21 00 00 00을 확인할 수 있었고, 리틀 엔디안 방식이므로 00 00 00 21 = 21임을 알 수 있다.

이는 21번째 block에 root directory file이 존재한다는 것을 의미한다.

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.



Block 0x21의 주소는 0x8400이다 (0x21 \* 400) 해당 위치로 이동하여 확인하였다.

[빨간색 : root]

Inode : 0200 0000 => 0000 0002 => 2 : inode 2번

Recode\_length : 0c00 => 000c => 12 : 파일전체 길이 12byte

Name\_length : 01 => 1 : 파일이름 길이 : 1byte

File\_type : 02 => 2 : root directory는 file type이 2

Name : 2e00 0000 => 0000 002e => .

[파란색 : 부모디렉토리 -> root]

Inode : 0200 0000 => 0000 0002 => 2 : inode 2번

Recode\_length : 0c00 => 000c => 12 : 파일전체 길이 12byte

Name\_length : 02 => 2 : 파일이름 길이 : 2byte

File\_type : 02 => 2 : root directory는 file type이 2

Name : 2e2e 0000 => 0000 2e2e => ..

[흰색 : lost+found]

Inode : 0b00 0000 => 0000 000b => 11 : inode 11번

Recode\_length : e803 => 03e8 => 1000 : 파일전체 길이 1000byte

Name\_length : 0a => 10 : 파일이름 길이 : 10byte

File\_type : 02 => 2 : root directory는 file type이 2

Name : 6c6f 7374 2b66 6f75 6e64 => 646e 756f 662b 7473 6f6c => lost+found

F1 파일을 생성하지 않았으므로 lost+found에 1000byte가 모두 할당된 모습이다.

11) Read the member files and confirm the contents. Lost+found의 block 번호인 22번에 가보면



다음과 같고,

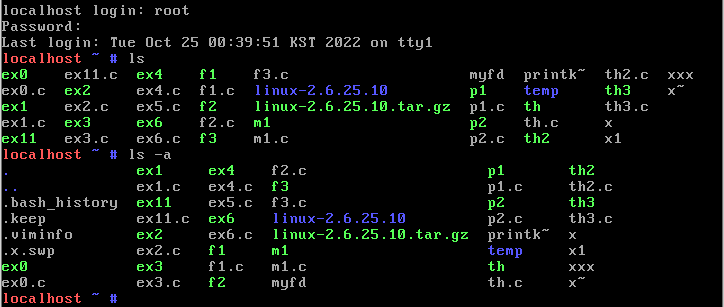
F1의 block 번호인 2f번에 가보면



다음과 같이 f1에 있는 내용을 볼 수 있다.

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.

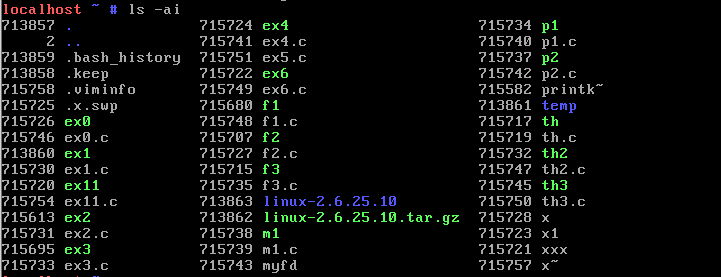
# ls -a



ls과 다르게 숨겨진 파일을 모두 파악할 수 있다.

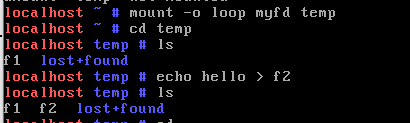
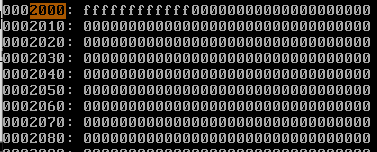
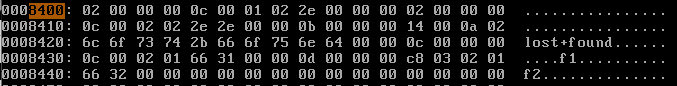
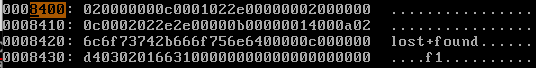
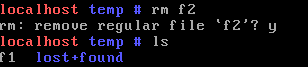
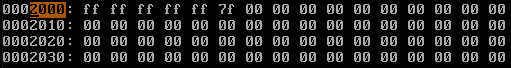
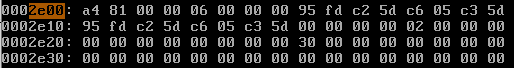
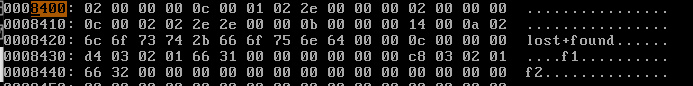
13) You can see the inode number of a file with "ls -i". Confirm the inode numbers of all files.

# ls -ai

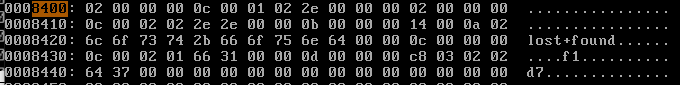


ls -ai는 inode number까지 출력하는 명령어이다.

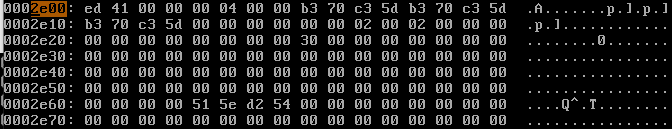
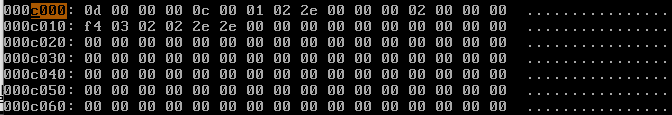
14) Make another file in your virtual disk. Confirm the changes in the file system: IBM, DBM, Inode table, and root directory file. Now delete this file (with "rm" command). What happens to the file system? How can you recover this file?

  
mount 한 후 새로운 파일 f2를 생성시켰다.  
변화를 살펴보기 위해 먼저 DBM 블락에 가보았다.  
DBM 블락은 2000h부터이다.  
  
  
7f가 ff로 바뀌었음을 알 수 있다. 추가하기 전에는 1번부터 47번까지의 블락을 사용했으나 이제는 48번까지 사용되고 있음을 알 수 있다.   
  
다음으로 IBM블락을 가보았다.  
  
  
  
  
ff 0f가 ff 1f로 바뀌었음을 알 수 있다.  
ff 0f는 1111 1111 / 0000 1111로서 1번부터 12번의 inode를 사용하고 있다는 것을 의미한다.   
ff 1f는 1111 1111 / 0001 1111로 1번부터 13번의 inode를 사용하고 있음을 의미한다.   
  
inode table에서 13번 inode로 가보았다.  
13번 inode는 2e00  
  
글자(hello)에 따라서 6바이트를 차지하고 블락 단위로는 2개의 블락을 차지한다.  
여기서의 블락단위는 512바이트이므로 실제로 1개의 블락이다.  
블락은 30, 즉 48번째 블락에 위치한다.  
  
  
루트 디렉토리 파일의 블락에 가보니 새롭게 f2에 관해 정의된 것을 볼 수 있다.  
  
  
이 전과 비교해서 f1에 관한 record length(3d4에서 12로)가 줄은 것을 알 수 있고 새롭게 f2에 대한 정보가 기록되어있다. f2는 inode number 0d=13, record length가 3c8, name length 2, file type은 data type으로 설정되어있다.  
  
  
이번에는 f2를 삭제해보았을 때 DBM, IBM, inode table에서 어떻게 변하는 지 살펴보도록 한다. f2는 48번 블락, 13번 inode에 해당하였다.  
  
  
DBM에서 48번 블락이 다시 사용하지 않는 것으로 바뀌었다.  
  
  
IBM에서도 13번 inode를 사용하지 않는 것으로 바뀌었다.  
  
inode table에서도 확인해보았다.  
  
  
inode table에 그대로 남아있다.  
  
  
root 디렉토리 파일 블락이다. root 디렉토리에서도 f2가 남아있다는 것을 알 수 있다.

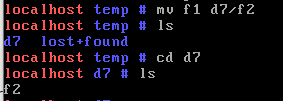
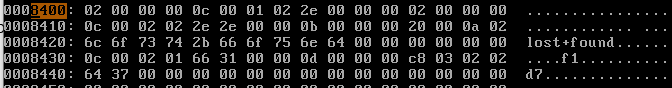
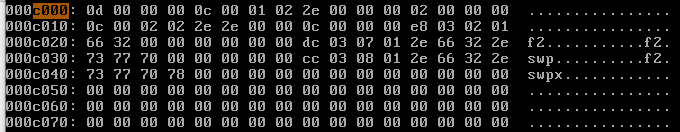
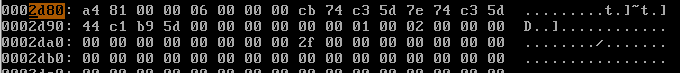
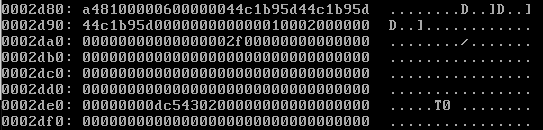
14-1) Make a new directory (d7) in the root directory (of myfd) with "mkdir" command. Show the disk block content of the root directory file and find out the inode number of d7.

  
temp에 d7이라는 디렉토리 파일을 추가해주었다.  
  
이 후에 root directory의 block내용을 살펴보았다.  
d7에 관한 정보가 담겨있다. d7의 inode number는 0d로 할당 되었다. 이는 decimal로 13번으로 되어있다. record length는 3c8, name length는 2, file type은 2번으로 directory file을 의미한다.

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?

13번 inode는 2e00이다. 해당 위치로 이동하여 살펴보았다.  
  
이로부터 d7의 바이트 사이즈는 400h = 1024, 블락사이즈는 2블락이므로, 1024블락 한 개를 차지한다. d7의 블락 위치는 30이다.  
  
  
d7의 블락이다. 이로부터 d7의 file content 중 첫번째는 inode number가 13번이고, record length가 12(decimal), name length는 1이고, file type이 2번으로 directory 파일이다.  
file name은 . 이다. 이로써 .은 현재 폴더이므로 d7과 같은 inode number를 갖는 것이다.  
다음 file은 inode number가 2번이고, record length는 3f4, name length는 2이고, file type는 directory file이다. file name은 .. 으로써 부모 디렉토리이다.  
즉 inode number 2번은 root directory이므로 ..은 root directory를 의미한다.

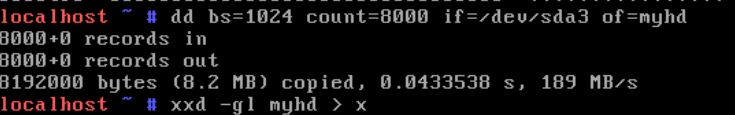
14-3) Run "mv f1 d7/f2" and show the changes in the root directory file, d7 file, and inode table.

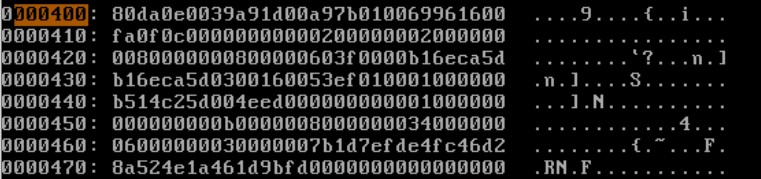
   
mv 명령어는 파일을 옮기는 것을 의미한다. mv f1 d7/f2 명령어는 f1파일을 d7 디렉토리 안으로 이동시키고 이름을 f2로 바꾼다는 것을 의미한다.  
  
  
root directory에 f1이 그대로 존재하고 inode number를 비롯한 정보들이 그대로 있다는 것을 알 수 있다. f1의 inode number는 12번이다.  
  
  
반면 d7 블락에 .과 ..을 제외하고 f2, f2.swp, f2.swpx 파일들이 생겼다는 것을 알 수 있다.  
f2의 inode number는 앞선 f1과 같은 12번이라는 것을 알 수 있다. 그러나 파일 이름은 f2로 나타내져있다.  
나머지 swp, swpx 파일은 찾아보니 스왑파일이라고 불린다. 스왑파일은 파일을 편집하다가 예기지 않게 종료해버린 경우 또는 다른 사용자가 먼저 해당 파일을 실행하고 있는 경우에 접근하려는 경우 생성된다고 한다. 이 경우는 mv 명령어를 사용하면서 생긴 경우나 아니면 본인이 vi 에디터를 제대로 종료하지 않아서 생긴 것으로 판단된다.  
  
  
  
  
  
대체로 같은 구성을 띄고 있으며 둘다 해당 블락이 2f라고 가르키고 있다.  
따라서 2f번째 블락을 살펴보았다.  
  
여전히 korea라는 내용을 띄고 있다는 것을 알 수 있다.  
  
이것으로 move로 위치를 바꿔도 inode table에서의 위치는 바뀌지 않고 블락의 위치 또한 바뀌지 않는다. 해당 directory 파일 블락에만 그 파일이 존재하는 것으로 바뀐다.  
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





00 0e da 80 // u32 m\_inodes\_count;

00 1d a9 39 // u32 m\_blocks\_count;

00 01 7b a9 // u32 m\_r\_blocks\_count;

00 16 af 90 // u32 m\_free\_blocks\_count;

00 0c 10 31 // u32 m\_free\_blocks\_count; super block의 위치가 0 이므로 group descriptor 의 위치는 1 번째 block 이다.

00 00 00 00 // u32 m\_first\_data\_block;

00 00 00 02 // u32 m\_log\_ block\_size; 1024\* (2^2) = 4KB = 1000h

Super block의 위치는 항상 400h로 일정하다.

Block\_size가 1000h이므로 1000h에 가보면



DBM block number : 00 00 01 dc = 476

IBM block number : 00 00 01 dd = 477

Inode table block number : 00 00 01 de = 478

DBM location : 00 1d c0 00

DBM location : 00 1d c0 00

IBM location : 00 1d d0 00

Inode table location : 00 1d e0 00

16) Write a program that opens a disk device formatted with EXT2 and reads and displays the super block, group descriptor, ibm, dbm, and inode table. Also display the file names in the root directory file, their inode numbers, and their block locations. Use open(), lseek(), read(), etc.

struct superblock{

int total\_inode\_num;

int total\_block\_num;

.........

};

int x; char buf[1024]; struct superblock \*sb;

x=open("myfd", O\_RDONLY, 00777); //open a virtual disk

lseek(x, 1024, SEEK\_SET); // move the file pointer to offset 1024 where the

// superblock starts

read(x, buf, 1024); // read the superblock into buf

sb=(struct superblock \*)buf; // interpret the data in buf as "struct superblock"

printf("total inode num:%x, total\_block\_num:%x, ...",

sb->total\_inode\_num, sb->total\_block\_num, ....);