2024 OS LAB3

Dankook University
Minguk Choi

Content

- Prepare for practice
- 1. File system layout analysis with Digital Forensic
- 2. File system modification with Module Programming
- Appendix

EXT2 file system 실습

File System Layout Analysis
with Digital Forensic

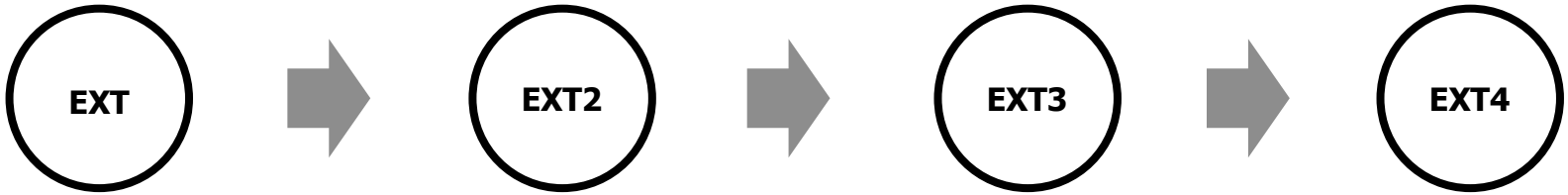
1. File system layout analysis

- Goals of OS Lab3
  - ✓ EXT2 file system에서 진행하는 Digital Forensic입니다
  - ✓ 자신의 학번 끝 세 자리에 해당하는 파일을 찾아가는 것이 과제입니다
    - 파일은 총 세 개의 블럭으로 구성되어있습니다
    - 여기서 한 블럭을 추가해서 총 네 개의 블럭을 찾으면 됩니다
    - 파일을 찾아가기까지의 모든 과정을 상세히 설명할 것 (이후에 나오는 예시와 같이)
  - ✓ 예시 답안) 자신의 학번이 32153550이라면 5번 디렉터리 안에 50번 파일을 찾아야합니다
    - 학생들이 최종적으로 찾아야 할 디스크 블록 번호와 그 내용 (실제 찾아야 할 번호는 다를 수 있음)

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x38426000 /dev/ramdisk
38426000: 352f3530 2d310a00 00000000 00000000  5/50-1..........
38426010: 00000000 00000000 00000000 00000000  ................

root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x10bc6000 /dev/ramdisk
10bc6000: 352f3530 2d320a00 00000000 00000000  5/50-2..........
10bc6010: 00000000 00000000 00000000 00000000  ................

root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x10c67000 /dev/ramdisk
10c67000: 352f3530 2d330a00 00000000 00000000  5/50-3..........
10c67010: 00000000 00000000 00000000 00000000  ................

root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x1102a000 /dev/ramdisk
1102a000: 352f3530 2d31330a 00000000 00000000  5/50-13.........
1102a010: 00000000 00000000 00000000 00000000  ................
```

1. File system layout analysis

- EXT2 file system
  - ✓ EXT2(extended file system)은 미닉스 파일시스템을 개선하여 만들어진 파일시스템이다.
  - ✓ EXT file system은 리눅스에서 일반 파일 API를 제공하기 위해 가상 파일 시스템 레이어(VFS)가 리눅스 커널에 추가되면서 만들어졌다.
  - ✓ EXT 파일 시스템은 1992년 첫 등장 이후에 문제점들을 자체적으로 개선해나가며 추후에 EXT2, EXT3 그리고 EXT4로 확장되었다.



1992년
VFS API를 최초로 활용한
파일 시스템

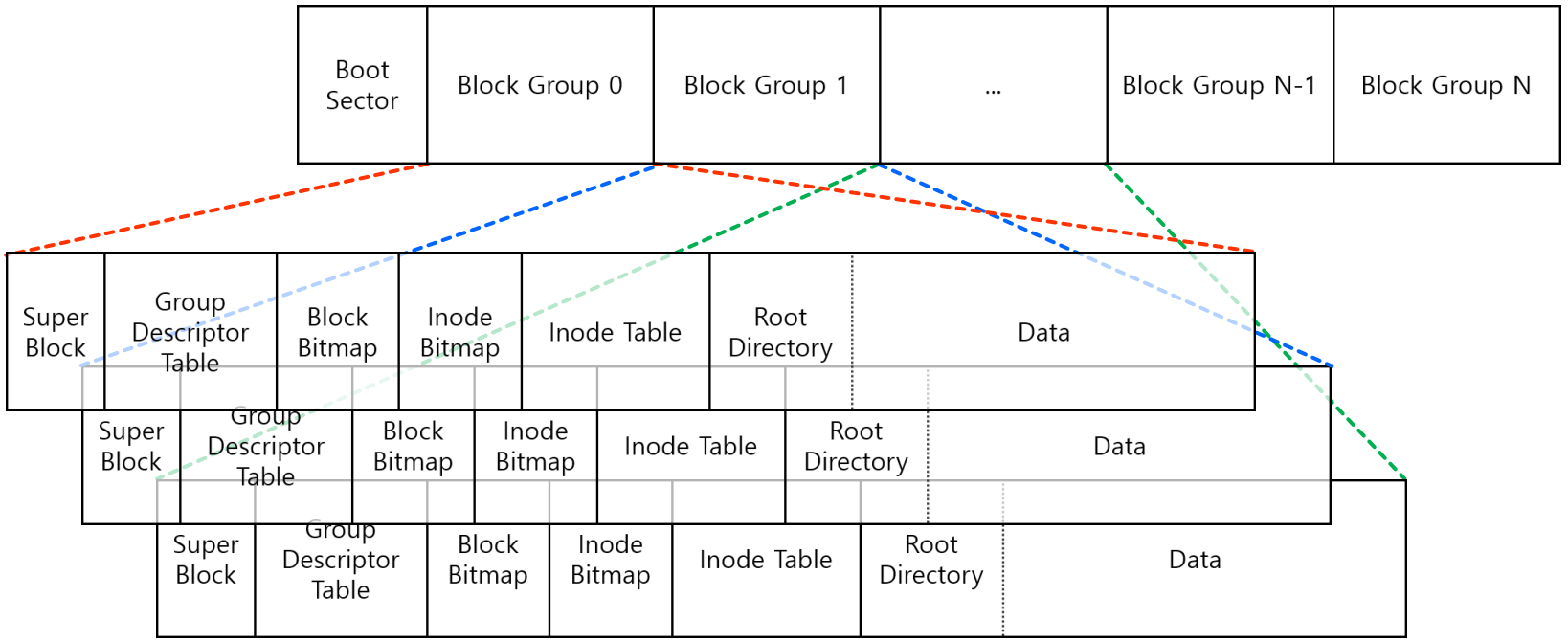
1993년
파일 접근 타임스탬프
데이터, 아이노드 수정 지원
블럭 그룹 도입

2001년
저널링 도입

2008년
Extent와 지연할당 도입
하위 디렉터리 제한 해제

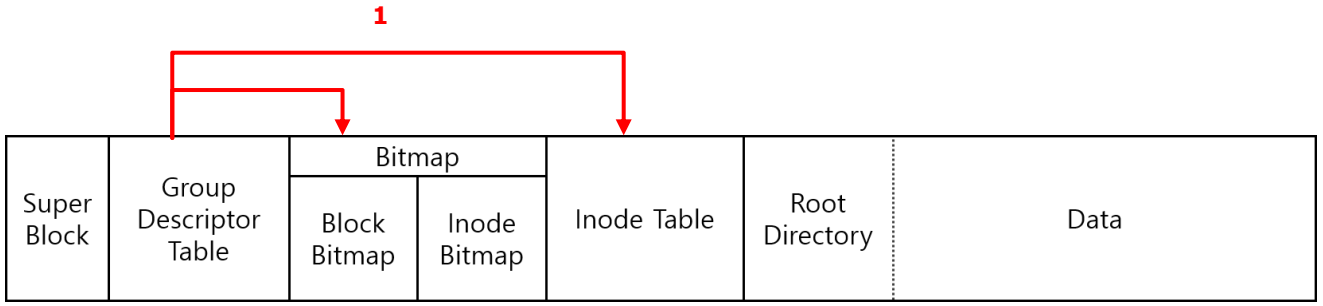
1. File system layout analysis

- EXT2 file system Layout
  - ✓ EXT2는 Boot Sector와 여러 개의 Block Group들로 구성
  - ✓ Boot Sector
    - 부팅을 위한 여러가지 값을 저장하고 있고, 크기는 2 sectors (1KB = 0x400) 이다.
  - ✓ Block Group
    - Ext2 File system은 인접한 Track들을 한개의 Block Group으로 묶어 관리한다 (FFS의 실린더 그룹).
    - Directory와 같이 논리적으로 연관성이 있는 Data를 한 Block Group 저장한다.
    - Super Block과 Group Descriptor Table 같은 중요한 영역을 중복해서 관리한다.



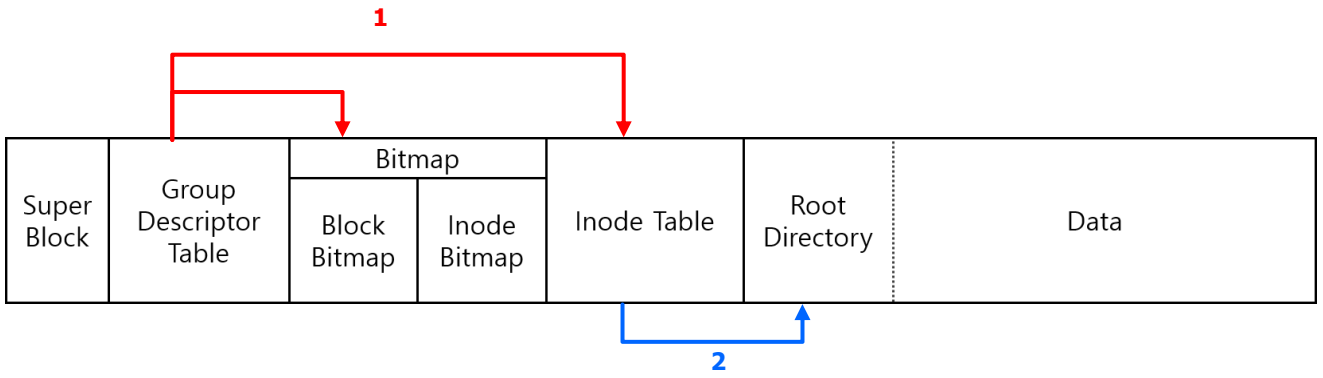
1. File system layout analysis

- EXT2 file system Layout
  - ✓ Super Block
    - 파일 시스템 전반적인 정보를 가지고있다.
    - Block count, Inode count, Blocks per group, Block Group count...
    - EXT2에서 슈퍼블럭은 각각의 블럭 그룹에 복제된 상태로 존재한다.
  - ✓ Group Descriptor Table
    - 각 Block Group에 대한 Description을 나열한 테이블이다.
    - Descriptor에는 Block Bitmap, Inode Bitmap, Inode Table이 포함되어있다.
  - ✓ Bitmap
    - 해당 Block이나 Inode Number가 할당되었는지 표시하는 비트맵이다.



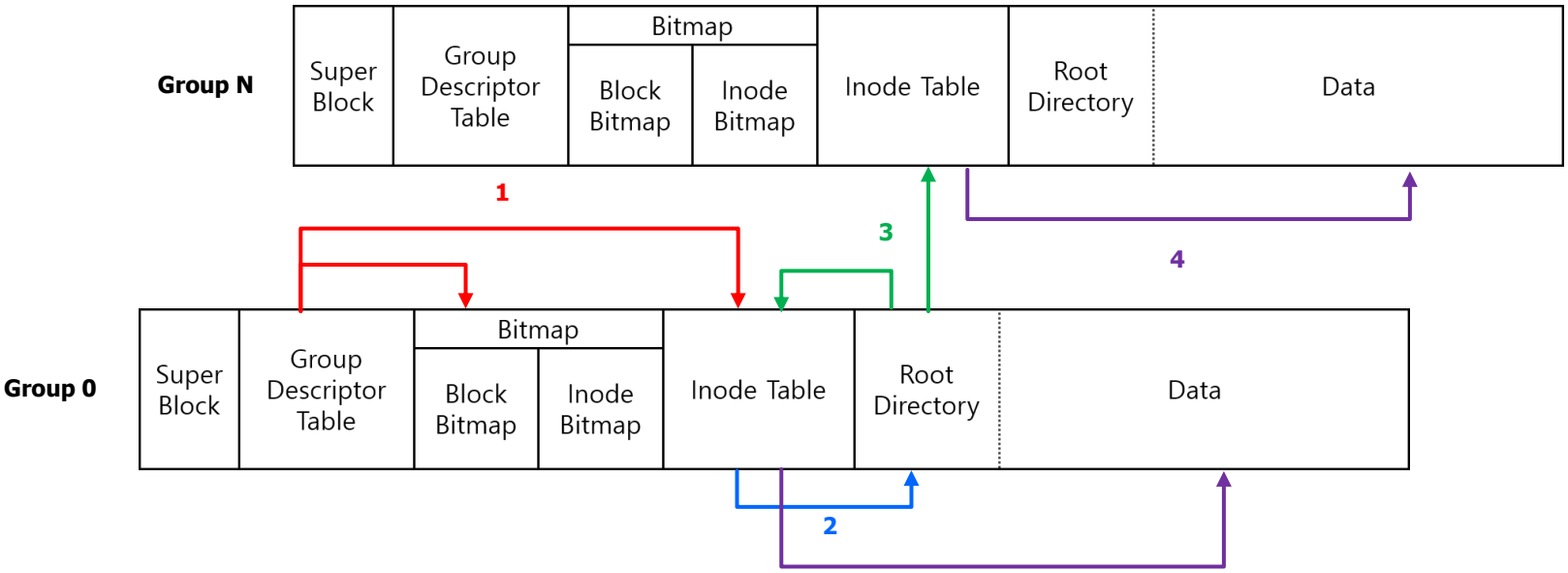
1. File system layout analysis

- EXT2 file system Layout
  - ✓ Inode Table
    - 해당 Block Group에 할당된 Inode들을 나열한 테이블이다.
    - Inode는 12개의 Direct Pointer와 3개의 Indirect Pointer로 구성되어있다.
    - 각 Inode의 크기는 256 byte이고 Root Directory의 아이노드 번호는 2번이다.
  - ✓ Data
    - 파일 혹은 디렉터리가 저장되는 공간이다.
    - EXT2에서는 반드시 Inode Table 뒤 영역에 Root Directory가 존재한다
    - 디렉터리는 그에 속한 파일이나 하위 디렉터리의 정보를 담은 Directory Entry의 Table이다.
    - 파일은 파일의 내용을 담고있다.



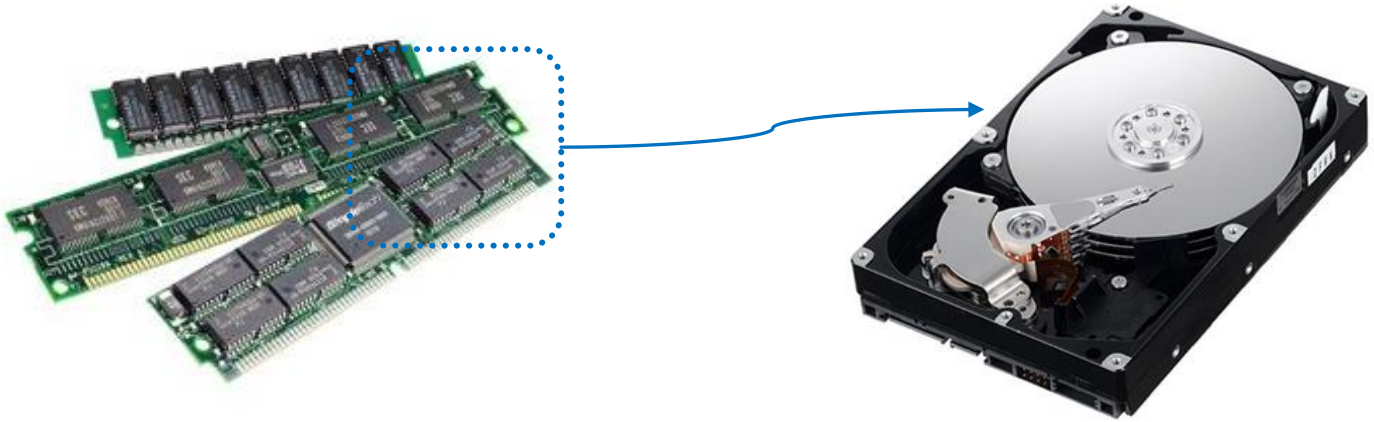
1. File system layout analysis

- EXT2 file system Layout
  - ✓ Data
    - Directory Entry
      - 하위 디렉터리나 파일의 Inode Number를 가지고 있다.
      - Inode Number 를 가지고 해당 그룹의 Inode Table에서 Inode를 찾아 접근할 수 있다.
      - EXT2는 논리적으로 연관성이 있는 Data를 한 Block Group 저장하려하지만 그렇지 못하는 경우도 있다.



1. File system layout analysis

- RAM Disk
  - ✓ RAM Disk는 주 기억 장치 활용 저장법이 아닌 RAM(DRAM)을 이용하여 디스크 드라이브를 구현하는 방식
  - ✓ 본 강의에서는 실제 Storage를 사용하지 않고 Ram Disk를 생성하여 이를 Storage로 활용한다.
  - ✓ 실습 자료로 공유된 이미지에는 쉽게 설치할 수 있도록 Ram Disk 설치 파일이 준비되어 있다.



1. File system layout analysis

- RAM Disk
  - ✓ ls 명령어로 모든 파일이 있는지 확인후 sudo su로 root 권한으로 변경
  - ✓ make
  - ✓ ramdisk.ko 파일이 있는지 확인

```
sys32153550@ESL-LeeJY:~/workspace/2020_1/OS_Lab3$ ls
append.c  create.sh  Makefile  ramdisk.c
sys32153550@ESL-LeeJY:~/workspace/2020_1/OS_Lab3$ sudo su
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# make
make -C /lib/modules/5.3.0-42-generic/build M=/home/sys32153550/workspace/2020_1/OS_Lab3 modules
make[1]: Entering directory '/usr/src/linux-headers-5.3.0-42-generic'
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/ramdisk.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/sys32153550/workspace/2020_1/OS_Lab3/ramdisk.mod.o
  LD [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/ramdisk.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.3.0-42-generic'
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ls
append.c  create.sh  Makefile  modules.order  Module.symvers  ramdisk.c  ramdisk.ko  ramdisk.mod  ramdisk.mod.c  ramdisk.mod.o  ramdisk.o
```

  - ✓ 모듈 적재 후 확인
  - ✓ insmod ramdisk.ko
  - ✓ lsmod | grep ramdisk

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# insmod ramdisk.ko
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# lsmod | grep ramdisk
ramdisk                16384  0
```

  - ✓ mkdir mnt

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# mkdir mnt
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ls
append.c  create.sh  Makefile  mnt  modules.order  Module.symvers  ramdisk.c  ramdisk.ko  ramdisk.mod  ramdisk.mod.c  ramdisk.mod.o  ramdisk.o
```

1. File system layout analysis

- RAM Disk
  - ✓ 파일시스템 포멧 후 mnt에 마운트
  - ✓ mkfs.ext2 /dev/ramdisk
  - ✓ mount /dev/ramdisk ./mnt

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# mkfs.ext2 /dev/ramdisk
mke2fs 1.44.1 (24-Mar-2018)
Creating filesystem with 262144 4k blocks and 65536 inodes
Filesystem UUID: 5f361a67-3aaf-48aa-9013-7e3ab1080ffd
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# mount /dev/ramdisk ./mnt
```

  - ✓ 잘 되었는지 확인
  - ✓ df –h

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev             12G     0   12G   0% /dev
tmpfs           2.4G  1.5M  2.4G   1% /run
/dev/sda1       469G   56G  390G  13% /
tmpfs            12G     0   12G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs            12G     0   12G   0% /sys/fs/cgroup
/dev/loop2       15M   15M     0 100% /snap/gnome-characters/495
/dev/loop3      1.0M  1.0M     0 100% /snap/gnome-logs/93
/dev/loop1      161M  161M     0 100% /snap/gnome-3-28-1804/116
/dev/loop0      4.4M  4.4M     0 100% /snap/gnome-calculator/704
/dev/ramdisk   1008M  1.3M  956M   1% /home/sys32153550/workspace/2020_1/OS_Lab3/mnt
```

1. File system layout analysis

- RAM Disk
  - ✓ ./create.sh
  - ✓ 스크립트 실행 후 mnt에 0~9번 디렉터리가 생성된 것을 확인할 수 있다.

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ./create.sh
create files ...
done
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ls mnt
0  1  2  3  4  5  6  7  8  9  lost+found
```

  - ✓ 각 디렉터리 안에는 파일이 0~99번까지 생성되어있다. (여기 까지는 한 파일 별로 3개의 블록)

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ls mnt/0
0   12  16  2   23  27  30  34  38  41  45  49  52  56  6   63  67  70  74  78  81  85  89  92  96
1   13  17  20  24  28  31  35  39  42  46  5   53  57  60  64  68  71  75  79  82  86  9   93  97
10  14  18  21  25  29  32  36  4   43  47  50  54  58  61  65  69  72  76  8   83  87  90  94  98
11  15  19  22  26  3   33  37  40  44  48  51  55  59  62  66  7   73  77  80  84  88  91  95  99
```

  - ✓ 여기서 자신이 찾아야 할 파일에 한 블럭을 추가해줄 것이다. (결국 4개의 블록이 됨)
  - ✓ ls -l mnt/5/50
  - ✓ ./apd mnt/5/50 13 5/50-13 (5, 50은 자신의 학번 마지막 세자리, 여기서 apd는 조교가 제공하는 도구임)
  - ✓ ls -l mnt/5/50 파일크기 49160 확인

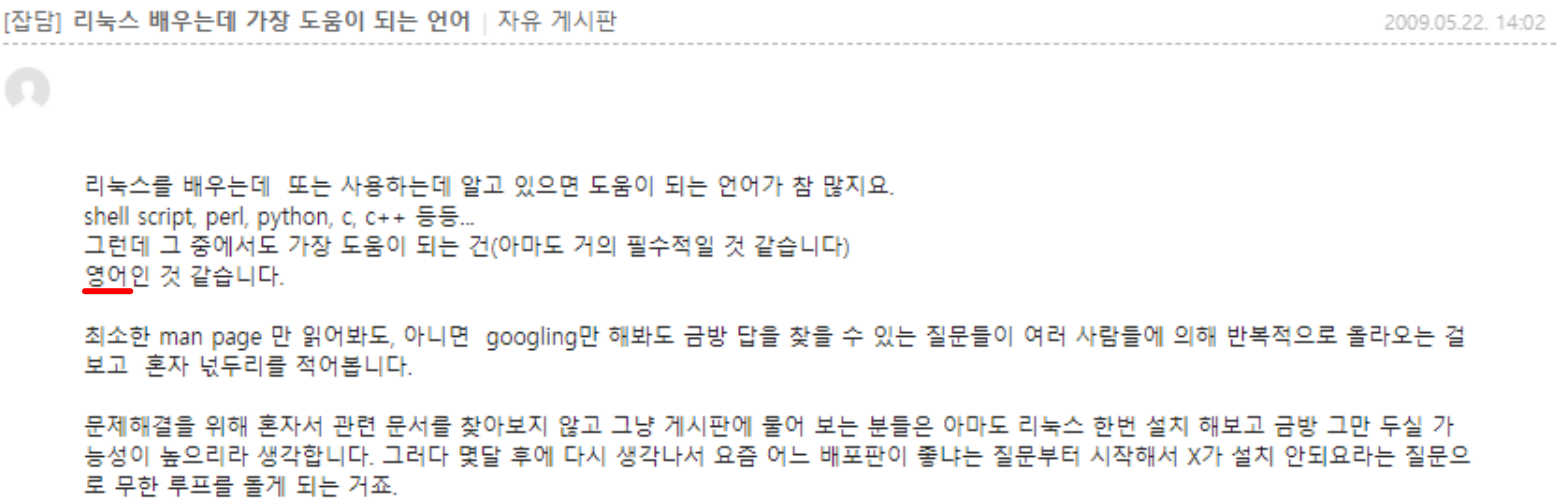
```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ls -l mnt/5/50
-rw-r--r-- 1 root root 8199  4월  21 15:50 mnt/5/50
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ./apd mnt/5/50 13 5/50-13
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ls -l mnt/5/50
-rw-r--r-- 1 root root 49160  4월  22 15:21 mnt/5/50
```

1. File system layout analysis

- EXT2 Hex Data 읽기
  - ✓ EXT2 의 정보는 리틀 엔디안으로 저장되있고, 일반 데이터는 빅엔디안으로 저장되있다.
  - ✓ 따라서, EXT2의 Metadata를 읽을 때는 보여지는 데이터를 바이트별로 반대로 읽어야한다.

0x	12	34	56	78
0x	78	56	34	12

- EXT2의 Data Structure와 다른 세부 정보는 아래 사이트에서 확인할 수 있다.
  - ✓ http://www.nongnu.org/ext2-doc/ext2.html#superblock



[잡담] 리눅스 배우는데 가장 도움이 되는 언어 | 자유 게시판 2009.05.22. 14:02

리눅스를 배우는데 또는 사용하는데 알고 있으면 도움이 되는 언어가 참 많지요.
shell script, perl, python, c, c++ 등등...
그런데 그 중에서도 가장 도움이 되는 건(아마도 거의 필수적일 것 같습니다)
영어인 것 같습니다.

최소한 man page 만 읽어봐도, 아니면 googling만 해봐도 금방 답을 찾을 수 있는 질문들이 여러 사람들에 의해 반복적으로 올라오는 걸 보고 혼자 넋두리를 적어봅니다.

문제해결을 위해 혼자서 관련 문서를 찾아보지 않고 그냥 게시판에 물어 보는 분들은 아마도 리눅스 한번 설치 해보고 금방 그만 두실 가능성이 높으리라 생각합니다. 그러다 몇달 후에 다시 생각나서 요즘 어느 배포판이 좋냐는 질문부터 시작해서 X가 설치 안되요라는 질문으로 무한 루프를 돌게 되는 거죠.

1. File system layout analysis

- Super Block 영역 분석

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x400 /dev/ramdisk
00000400: 00000100 00000400 33330000 a5ee0300  ........33......
00000410: f5ff0000 00000000 02000000 02000000  ................
00000420: 00800000 00800000 00200000 cb7a9d5e  ......... ...z.^
00000430: cb7a9d5e 0100ffff 53ef0000 01000000  .z.^....S.......
00000440: c57a9d5e 00000000 00000000 01000000  .z.^............
00000450: 00000000 0b000000 00010000 38000000  ............8...
00000460: 02000000 03000000 81f90f30 853f4530  ...........0.?E0
00000470: a4c72cbe 9b2a3d79 00000000 00000000  ..,..*=y........
00000480: 00000000 00000000 2f686f6d 652f7379  ......../home/sy
00000490: 73333231 35333535 302f776f 726b7370  s32153550/worksp
000004a0: 6163652f 32303230 5f312f4f 535f4c61  ace/2020_1/OS_La
000004b0: 62332f6d 6e740000 00000000 00000000  b3/mnt..........
000004c0: 00000000 00000000 00000000 00003f00  ..............?.
000004d0: 00000000 00000000 00000000 00000000  ................
000004e0: 00000000 00000000 00000000 eb942176  ..............!v
000004f0: 16dc40dd 8182758f b08f718d 01000000  ..@...u...q.....
```

- xxd(hexdump) [options] [infile]
  - ✓ infile의 데이터를 16진수 형태로 보여준다
  - ✓ options
    - -l len : len만큼 데이터를 읽는다
    - -s [+/-] offset : offset만큼 오프셋을 이동
      - +/- : lseek()의 SEEK_SET/SEEK_END와 같음
    - -g bytes : bytes만큼 묶어서 출력한다.
  - ✓ ex) xxd –g 4 –l 0x100 –s 0x400 /dev/ramdisk
  - ✓ /dev/ramdisk의 0x400 byte부터 0x100 byte만큼 4 bytes 씩 묶어서 보여준다.
  - ✓ 추가적인 옵션은 아래 사이트에서 확인가능
  - ✓ https://linux.die.net/man/1/xxd

1. File system layout analysis

- Super Block 영역 분석

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x400 /dev/ramdisk
00000400: 00000100 00000400 33330000 a5ee0300  ........33......
00000410: f5ff0000 00000000 02000000 02000000  ................
00000420: 00800000 00800000 00200000 cb7a9d5e  ......... ...z.^
00000430: cb7a9d5e 0100ffff 53ef0000 01000000  .z.^....S.......
00000440: c57a9d5e 00000000 00000000 01000000  .z.^............
00000450: 00000000 0b000000 00010000 38000000  ............8...
00000460: 02000000 03000000 81f90f30 853f4530  ...........0.?E0
00000470: a4c72cbe 9b2a3d79 00000000 00000000  ..,..*=y........
00000480: 00000000 00000000 2f686f6d 652f7379  ......../home/sy
00000490: 73333231 35333535 302f776f 726b7370  s32153550/worksp
000004a0: 6163652f 32303230 5f312f4f 535f4c61  ace/2020_1/OS_La
000004b0: 62332f6d 6e740000 00000000 00000000  b3/mnt..........
000004c0: 00000000 00000000 00000000 00003f00  ..............?.
000004d0: 00000000 00000000 00000000 00000000  ................
000004e0: 00000000 00000000 00000000 eb942176  ..............!v
000004f0: 16dc40dd 8182758f b08f718d 01000000  ..@...u...q.....
```

```
/*
 * Structure of the super block
 */
struct ext2_super_block {
        __le32  s_inodes_count;         /* Inodes count */
        __le32  s_blocks_count;         /* Blocks count */
        __le32  s_r_blocks_count;       /* Reserved blocks count */
        __le32  s_free_blocks_count;    /* Free blocks count */
        __le32  s_free_inodes_count;    /* Free inodes count */
        __le32  s_first_data_block;     /* First Data Block */
        __le32  s_log_block_size;       /* Block size */
        __le32  s_log_frag_size;        /* Fragment size */
        __le32  s_blocks_per_group;     /* # Blocks per group */
        __le32  s_frags_per_group;      /* # Fragments per group */
        __le32  s_inodes_per_group;     /* # Inodes per group */
        __le32  s_mtime;                /* Mount time */
        __le32  s_wtime;                /* Write time */
        __le16  s_mnt_count;            /* Mount count */
        __le16  s_max_mnt_count;        /* Maximal mount count */
        __le16  s_magic;                /* Magic signature */
        __le16  s_state;                /* File system state */
        __le16  s_errors;               /* Behaviour when detecting errors */
        __le16  s_minor_rev_level;      /* minor revision level */
```

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	inode count				block count				res block count				free block count			
10	free inode count				first data block				log block size				log frag size			
20	block per group				frag per group				inode per group				mtime			
30	wtime				mount count		max mount size		magic		state		errors		minor version	
40	last check				check interval				creator OS				major version			
50	def_res uid		def_res gid		first non-reserved inode				inode size		block grp num		compatible feature flag			
60	incompatible feature flag				feature read only compat				uuid (16 byte)							
70									volume name (16 byte)							
80																
90	last mounted (64 byte)												prealloc dir block			
a0																
b0													prealloc block			
c0									algorithm usage bitmap						padding	
d0	journal uuid															
e0	journal inode number				journal device				last orphan							
f0	hash seed (16 byte)													pad	padding	
100	default mount option				first meta block								default hash version			

1. File system layout analysis

- Super Block 영역 분석

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x400 /dev/ramdisk
00000400: 00000100 00000400 33330000 a5ee0300  ........33......
00000410: f5ff0000 00000000 02000000 02000000  ................
00000420: 00800000 00800000 00200000 cb7a9d5e  ......... ...z.^
00000430: cb7a9d5e 0100ffff 53ef0000 01000000  .z.^....S.......
00000440: c57a9d5e 00000000 00000000 01000000  .z.^............
00000450: 00000000 0b000000 00010000 38000000  ............8...
00000460: 02000000 03000000 81f90f30 853f4530  ...........0.?E0
00000470: a4c72cbe 9b2a3d79 00000000 00000000  ..,..*=y........
00000480: 00000000 00000000 2f686f6d 652f7379  ......../home/sy
00000490: 73333231 35333535 302f776f 726b7370  s32153550/worksp
000004a0: 6163652f 32303230 5f312f4f 535f4c61  ace/2020_1/OS_La
000004b0: 62332f6d 6e740000 00000000 00000000  b3/mnt..........
000004c0: 00000000 00000000 00000000 00003f00  ..............?.
000004d0: 00000000 00000000 00000000 00000000  ................
000004e0: 00000000 00000000 00000000 eb942176  ..............!v
000004f0: 16dc40dd 8182758f b08f718d 01000000  ..@...u...q.....
```

- ✓ inode count : 0x10000
- ✓ block count : 0x40000
- ✓ log block size : 0x2
- ✓ blocks per group : 0x8000
- ✓ inodes per group : 0x2000
- ✓ block group number : 0x0

※ 이는 예제로 실제 과제 내용과는 다름
(예제는 1GB 크기이나 과제는 512MB 크기의 램디스크를 사용하기 때문에 위 숫자 차이 있음)

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	inode count				block count				res block count				free block count			
10	free inode count				first data block				log block size				log frag size			
20	block per group				frag per group				inode per group				mtime			
30	wtime				mount count		max mount size		magic		state		errors		minor version	
40	last check				check interval				creator OS				major version			
50	def_res uid		def_res gid		first non-reserved inode				inode size		block grp num		compatible feature flag			
60	incompatible feature flag				feature read only compat				uuid (16 byte)							
70									volume name (16 byte)							
80																
90	last mounted (64 byte)												prealloc dir block			
a0																
b0													prealloc block			
c0									algorithm usage bitmap							padding
d0	journal uuid															
e0	journal inode number				journal device				last orphan							
f0	hash seed (16 byte)													pad		padding
100	default mount option				first meta block								default hash version			

1. File system layout analysis

- Group Descriptor Table 영역 분석
  - ✓ 첫번째 Group Descriptor Table은 램디스크의 1블록 이후부터 시작 (1블록 = 4KB = 0x1000)

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x100 -s 0x1000 /dev/ramdisk
00001000: 41000000 42000000 43000000 b67d901f  A...B...C....}..
00001010: 03000400 00000000 00000000 00000000  ................
00001020: 41800000 42800000 43800000 bc7d9b1f  A...B...C....}..
00001030: 01000400 00000000 00000000 00000000  ................
00001040: 00000100 01000100 02000100 45749b1f  ............Et..
00001050: 01000400 00000000 00000000 00000000  ................
00001060: 41800100 42800100 43800100 bb7d361f  A...B...C....}6.
00001070: 02000400 00000000 00000000 00000000  ................
00001080: 00000200 01000200 02000200 fc7d361f  ............}6.
00001090: 02000400 00000000 00000000 00000000  ................
000010a0: 41800200 42800200 43800200 bc7d9b1f  A...B...C....}..
000010b0: 01000400 00000000 00000000 00000000  ................
000010c0: 00000300 01000300 02000300 fd7d9b1f  ............}..
000010d0: 01000400 00000000 00000000 00000000  ................
000010e0: 41800300 42800300 43800300 bc7b9b1f  A...B...C....{..
000010f0: 01000400 00000000 00000000 00000000  ................
```

```
/*
 * Structure of a blocks group descriptor
 */
struct ext2_group_desc
{
        __le32  bg_block_bitmap;                /* Blocks bitmap block */
        __le32  bg_inode_bitmap;                /* Inodes bitmap block */
        __le32  bg_inode_table;         /* Inodes table block */
        __le16  bg_free_blocks_count;   /* Free blocks count */
        __le16  bg_free_inodes_count;   /* Free inodes count */
        __le16  bg_used_dirs_count;     /* Directories count */
        __le16  bg_pad;
        __le32  bg_reserved[3];
};
```

  - ✓ Group 0
    - block bitmap : 0x41 블록 부터 시작
    - Inode bitmap : 0x42 블록 부터 시작
    - Inode table : 0x43 블록 부터 시작
    - 이때 단위는 블럭(4KB)임을 감안해야한다.
  - ✓ inode가 속한 그룹은 (inode number – 1) / block per group 번 Block Group이다.
  - ✓ Ext2에서 Root inode number는 2번이다.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	block bitmap				inode bitmap				inode table				free blk cnt		free ino cnt	
10	used dir cnt		padding		reserved (padding)											

  - ✓ 예시에서 inodes per group는 0x2000이므로
    - root's block group : (2 – 1) / 0x2000 = 0
    - root's index : (2 – 1) % 0x2000 = 1
    - 0번 Block Group의 Inode Table의 1번째에 위치한다

1. File system layout analysis

- Inode Table 영역 분석

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x1000 -s 0x43000 /dev/ramdisk
00043000: 00000000 00000000 76619e5e 76619e5e  ........va.^va.^
00043010: 76619e5e 00000000 00000000 00000000  va.^............
00043020: 00000000 00000000 00000000 00000000  ................
00043030: 00000000 00000000 00000000 00000000  ................
00043040: 00000000 00000000 00000000 00000000  ................
00043050: 00000000 00000000 00000000 00000000  ................
00043060: 00000000 00000000 00000000 00000000  ................
00043070: 00000000 00000000 00000000 00000000  ................
00043080: 00000000 00000000 00000000 00000000  ................
00043090: 00000000 00000000 00000000 00000000  ................
000430a0: 00000000 00000000 00000000 00000000  ................
000430b0: 00000000 00000000 00000000 00000000  ................
000430c0: 00000000 00000000 00000000 00000000  ................
000430d0: 00000000 00000000 00000000 00000000  ................
000430e0: 00000000 00000000 00000000 00000000  ................
000430f0: 00000000 00000000 00000000 00000000  ................
00043100: ed410000 00100000 76619e5e 83619e5e  .A......va.^.a.^
00043110: 83619e5e 00000000 00000d00 08000000  .a.^............
00043120: 00000000 0a000000 43020000 00000000  ........C.......
00043130: 00000000 00000000 00000000 00000000  ................
00043140: 00000000 00000000 00000000 00000000  ................
00043150: 00000000 00000000 00000000 00000000  ................
00043160: 00000000 00000000 00000000 00000000  ................
00043170: 00000000 00000000 00000000 00000000  ................
00043180: 20000000 00f4bf88 00f4bf88 00000000   ...............
00043190: 76619e5e 00000000 00000000 00000000  va.^............
000431a0: 00000000 00000000 00000000 00000000  ................
000431b0: 00000000 00000000 00000000 00000000  ................
000431c0: 00000000 00000000 00000000 00000000  ................
000431d0: 00000000 00000000 00000000 00000000  ................
000431e0: 00000000 00000000 00000000 00000000  ................
000431f0: 00000000 00000000 00000000 00000000  ................
```

- ✓ Inode의 크기는 0x100 byte 이다 (inode + padding)
- ✓ Root가 속한 Block Group은 0번이고 Index는 1이므로
- ✓ 0번 Block Group의 Inode Table의 0x100 부터가 Root Inode이다

1. File system layout analysis

- Inode Table 영역 분석

```
100: ed410000 00100000 76619e5e 83619e5e  .A......va.^.a.^
110: 83619e5e 00000000 00000d00 08000000  .a.^............
120: 00000000 0a000000 43020000 00000000  ........C.......
130: 00000000 00000000 00000000 00000000  ................
140: 00000000 00000000 00000000 00000000  ................
150: 00000000 00000000 00000000 00000000  ................
160: 00000000 00000000 00000000 00000000  ................
170: 00000000 00000000 00000000 00000000  ................
180: 20000000 00f4bf88 00f4bf88 00000000   ...............
190: 76619e5e 00000000 00000000 00000000  va.^............
1a0: 00000000 00000000 00000000 00000000  ................
1b0: 00000000 00000000 00000000 00000000  ................
1c0: 00000000 00000000 00000000 00000000  ................
1d0: 00000000 00000000 00000000 00000000  ................
1e0: 00000000 00000000 00000000 00000000  ................
1f0: 00000000 00000000 00000000 00000000  ................
```

✓ mode : 0x41ed = drwxr-xr-x

Constant	Value	Description
-- file format --		
EXT2_S_IFSOCK	0xC000	socket
EXT2_S_IFLNK	0xA000	symbolic link
EXT2_S_IFREG	0x8000	regular file
EXT2_S_IFBLK	0x6000	block device
EXT2_S_IFDIR	0x4000	directory
EXT2_S_IFCHR	0x2000	character device
EXT2_S_IFIFO	0x1000	fifo
-- process execution user/group override --		
EXT2_S_ISUID	0x0800	Set process User ID
EXT2_S_ISGID	0x0400	Set process Group ID
EXT2_S_ISVTX	0x0200	sticky bit
-- access rights --		
EXT2_S_IRUSR	0x0100	user read
EXT2_S_IWUSR	0x0080	user write
EXT2_S_IXUSR	0x0040	user execute
EXT2_S_IRGRP	0x0020	group read
EXT2_S_IWGRP	0x0010	group write
EXT2_S_IXGRP	0x0008	group execute
EXT2_S_IROTH	0x0004	others read
EXT2_S_IWOTH	0x0002	others write
EXT2_S_IXOTH	0x0001	others execute

✓ block pointer 0 : 0x243 block

```
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        __le16  i_mode;         /* File mode */
        __le16  i_uid;          /* Low 16 bits of Owner Uid */
        __le32  i_size;         /* Size in bytes */
        __le32  i_atime;        /* Access time */
        __le32  i_ctime;        /* Creation time */
        __le32  i_mtime;        /* Modification time */
        __le32  i_dtime;        /* Deletion Time */
        __le16  i_gid;          /* Low 16 bits of Group Id */
        __le16  i_links_count;  /* Links count */
        __le32  i_blocks;       /* Blocks count */
        __le32  i_flags;        /* File flags */
        union {
                struct {
                        __le32  l_i_reserved1;
                } linux1;
                struct {
                        __le32  h_i_translator;
                } hurd1;
                struct {
                        __le32  m_i_reserved1;
                } masix1;
        } osd1;                         /* OS dependent 1 */
        __le32  i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
        __le32  i_generation;   /* File version (for NFS) */
        __le32  i_file_acl;     /* File ACL */
        __le32  i_dir_acl;      /* Directory ACL */
        __le32  i_faddr;        /* Fragment address */
```

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	mode			uid			size				access time				change time	
10	modification time					deletion time				gid		link count		blocks		
20	flags					OS description 1										
30	block pointer (60 byte)															
40																
50													**Indirect Pointer**			
60						generation				file access control list				dir access control list		
70	fragmentation blk addr					OS description 2										

1. File system layout analysis

- Data 영역 분석

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# xxd -g 4 -l 0x1000 -s 0x243000 /dev/ramdisk
00243000: 02000000 0c000102 2e000000 02000000  ................
00243010: 0c000202 2e2e0000 0b000000 14000a02  ................
00243020: 6c6f7374 2b666f75 6e640000 01400000  lost+found...@..
00243030: 0c000102 30000000 01c00000 0c000102  ....0...........
00243040: 31000000 01600000 0c000102 32000000  1....`......2...
00243050: 01800000 0c000102 33000000 01200000  ........3.... ..
00243060: 0c000102 34000000 01e00000 0c000102  ....4...........
00243070: 35000000 01a00000 0c000102 36000000  5...........6...
00243080: 0c000000 0c000102 37000000 66600000  ........7...f`..
00243090: 0c000102 38000000 66800000 680f0102  ....8...f...h...
002430a0: 39000000 00000000 00000000 00000000  9...............
```

- ✓ 5번 Directory
  - inode number : 0xe001
  - file type : 0x2 = directory

Constant Name	Value	Description
EXT2_FT_UNKNOWN	0	Unknown File Type
EXT2_FT_REG_FILE	1	Regular File
EXT2_FT_DIR	2	Directory File
EXT2_FT_CHRDEV	3	Character Device
EXT2_FT_BLKDEV	4	Block Device
EXT2_FT_FIFO	5	Buffer File
EXT2_FT_SOCK	6	Socket File
EXT2_FT_SYMLINK	7	Symbolic Link

  - 속한 Block Group : (0xe001 – 1) / 2000 = 7번 Block Group
  - Inode Table Index : (0xe001 – 1) % 2000 = 0
  - 5번 디렉터리의 Inode는 7번 Block Group의 Inode Table에서 0번째에 위치함을 알 수 있다.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	inode				record len		name len	file type	name (~255 byte)							

1. File system layout analysis

- 파일 찾기
  - ✓ 파일 찾기 순서는 다음과 같다.
    - Root Directory에서 찾을 File이 속한 Directory의 Inode Number를 찾는다 ← 예제의 과정
    - Diretory가 속한 Block Group의 Inode Table에서 Inode를 찾는다
    - 찾은 Directory Entry에서 File의 Inode Number를 찾는다
    - File이 속한 Block Group의 Inode Table에서 Inode를 찾는다
      - Ext2의 특성상 같은 Block Group에 할당되었을 가능성이 크다
  - ✓ 이전의 예시들과 같이 파일을 찾아가는 과정을 세세히 보여야한다

※ Ramdisk의 특성상 시스템을 종료할 경우 내용이 모두 지워지므로 참고할 것 !

EXT2 file system 실습 Bouns

File System Modification
with Module Programming

2. File system Modification

- EXT2 Filesystem Mount, lookup 시 자신의 이름 출력
  - ✓ 첫 번째 실습에서 mnt를 사용하고 있기 때문에 이를 정리한다.
  - ✓ umount /dev/ramdisk
  - ✓ rmmod ramdisk
  - ✓ lsmod | grep ramdisk (모듈이 없어졌는지 확인)

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# umount /dev/ramdisk
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# rmmod ramdisk
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# lsmod | grep ramdisk
```

  - ✓ insmod ramdisk.ko
  - ✓ lsmod | grep ramdisk

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# insmod ramdisk.ko
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# lsmod | grep ramdisk
ramdisk                16384  0
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3#
```

  - ✓ cd os_ext2
  - ✓ ls

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# cd os_ext2/
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2# ls
acl.c  balloc.c  ext2.h  ialloc.c  ioctl.c  Makefile  super.c    tags     xattr.h           xattr_trusted.c
acl.h  dir.c     file.c  inode.c   Kconfig  namei.c   symlink.c  xattr.c  xattr_security.c  xattr_user.c
```

  - ✓ 소스를 수정하여 마운트 시 자신의 이름이 출력되도록하면 성공
  - ✓ 출력 문구는 “os_ext2 : (자신의 이름) OS Lab3” 로 한다

```
[2510332.298832] os_ext2 : Lee Jeyeon OS Lab3
```

2. File system Modification

- Super Block
  - ✓ 마운트 된 파일 시스템에 대한 정보를 가진 객체
  - ✓ 파일 시스템 제어 블록(Filesystem control block)에 대응한다.

```
struct super_block {
        struct list_head        s_list;         /* Keep this first */
        dev_t                   s_dev;          /* search index; _not_ kdev_t */
        unsigned char           s_blocksize_bits;
        unsigned long           s_blocksize;
        loff_t                  s_maxbytes;     /* Max file size */
        struct file_system_type *s_type;
        const struct super_operations   *s_op;
        const struct dquot_operations   *dq_op;
        const struct quotactl_ops       *s_qcop;
        const struct export_operations *s_export_op;
        unsigned long           s_flags;
        unsigned long           s_iflags;       /* internal SB_I_* flags */
        unsigned long           s_magic;
        struct dentry           *s_root;
        struct rw_semaphore     s_umount;
        int                     s_count;
        atomic_t                s_active;
#ifdef CONFIG_SECURITY
        void                    *s_security;
#endif
        const struct xattr_handler **s_xattr;

        const struct fscrypt_operations *s_cop;

        struct hlist_bl_head    s_anon;         /* anonymous dentries for (nfs) exporting */
        struct list_head        s_mounts;       /* list of mounts; _not_ for fs use */
        struct block_device     *s_bdev;
        struct backing_dev_info *s_bdi;
        struct mtd_info         *s_mtd;
        struct hlist_node       s_instances;
        unsigned int            s_quota_types;  /* Bitmask of supported quota types */
        struct quota_info       s_dquot;        /* Diskquota specific options */

        struct sb_writers       s_writers;

        char                    s_id[32];       /* Informational name */
        uuid_t                  s_uuid;         /* UUID */

        void                    *s_fs_info;     /* Filesystem private info */
        unsigned int            s_max_links;
        fmode_t                 s_mode;
                                                                        1250,1
```

< include/linux/fs.h >

2. File system Modification

- Super Block
  - ✓ Super Block의 초기화는 파일 시스템이 mount 되는 시점에 수행된다.

```
static struct file_system_type ext2_fs_type = {
        .owner          = THIS_MODULE,
        .name           = "os_ext2",
        .mount          = ext2_mount,
        .kill_sb        = kill_block_super,
        .fs_flags       = FS_REQUIRES_DEV,
};
MODULE_ALIAS_FS("ext2");
```

```
static struct dentry *ext2_mount(struct file_system_type *fs_type,
        int flags, const char *dev_name, void *data)
{
        return mount_bdev(fs_type, flags, dev_name, data, ext2_fill_super);
}
```

```
static int ext2_fill_super(struct super_block *sb, void *data, int silent)
{
        struct dax_device *dax_dev = fs_dax_get_by_bdev(sb->s_bdev);
        struct buffer_head * bh;
        struct ext2_sb_info * sbi;
        struct ext2_super_block * es;
        struct inode *root;
        unsigned long block;
        unsigned long sb_block = get_sb_block(&data);
        unsigned long logic_sb_block;
        unsigned long offset = 0;
        unsigned long def_mount_opts;
```

2. File system Modification

- Super Block
  - ✓ ext2_fill_super 함수에서는 Disk의 super block을 읽고 ext2_boot_sector(boot record), superblock(ext2_sb_info), root directory(inode) 등의 정보를 초기화한다.

superblock 초기화

```
set_opt(opts.s_mount_opt, RESERVATION);

if (!parse_options((char *) data, sb, &opts))
        goto failed_mount;

sbi->s_mount_opt = opts.s_mount_opt;
sbi->s_resuid = opts.s_resuid;
sbi->s_resgid = opts.s_resgid;

sb->s_flags = (sb->s_flags & ~SB_POSIXACL) |
        ((EXT2_SB(sb)->s_mount_opt & EXT2_MOUNT_POSIX_ACL) ?
         SB_POSIXACL : 0);
sb->s_iflags |= SB_I_CGROUPWB;
```

root 생성

```
root = ext2_iget(sb, EXT2_ROOT_INO);
if (IS_ERR(root)) {
        ret = PTR_ERR(root);
        goto failed_mount3;
}
if (!S_ISDIR(root->i_mode) || !root->i_blocks || !root->i_size) {
        iput(root);
        ext2_msg(sb, KERN_ERR, "error: corrupt root inode, run e2fsck");
        goto failed_mount3;
}

sb->s_root = d_make_root(root);
```

superblock 읽기

```
/*
 * If the superblock doesn't start on a hardware sector boundary,
 * calculate the offset.
 */
if (blocksize != BLOCK_SIZE) {
        logic_sb_block = (sb_block*BLOCK_SIZE) / blocksize;
        offset = (sb_block*BLOCK_SIZE) % blocksize;
} else {
        logic_sb_block = sb_block;
}

if (!(bh = sb_bread(sb, logic_sb_block))) {
        ext2_msg(sb, KERN_ERR, "error: unable to read superblock");
        goto failed_sbi;
}
```

ext2_sb_info 초기화

```
sbi->s_frag_size = EXT2_MIN_FRAG_SIZE <<
                        le32_to_cpu(es->s_log_frag_size);
if (sbi->s_frag_size == 0)
        goto cantfind_ext2;
sbi->s_frags_per_block = sb->s_blocksize / sbi->s_frag_size;

sbi->s_blocks_per_group = le32_to_cpu(es->s_blocks_per_group);
sbi->s_frags_per_group = le32_to_cpu(es->s_frags_per_group);
sbi->s_inodes_per_group = le32_to_cpu(es->s_inodes_per_group);

sbi->s_inodes_per_block = sb->s_blocksize / EXT2_INODE_SIZE(sb);
```

2. File system Modification

- EXT2 Filesystem Mount, lookup 시 자신의 이름 출력
  - ✓ 소스 수정 후 make
  - ✓ make

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2# make
make -C /lib/modules/5.3.0-42-generic/build M=/home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.3.0-42-generic'
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/balloc.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/dir.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/file.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/ialloc.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/inode.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/ioctl.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/namei.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/super.o
  CC [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/symlink.o
  LD [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/os_ext2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/os_ext2.mod.o
  LD [M]  /home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2/os_ext2.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.3.0-42-generic'
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2# ls
acl.c     dir.c   file.o    inode.o  Makefile       namei.o        os_ext2.mod.o  symlink.c  xattr.h
acl.h     dir.o   ialloc.c  ioctl.c  modules.order  os_ext2.ko     os_ext2.o      symlink.o  xattr_security.c
balloc.c  ext2.h  ialloc.o  ioctl.o  Module.symvers  os_ext2.mod    super.c        tags       xattr_trusted.c
balloc.o  file.c  inode.c   Kconfig  namei.c        os_ext2.mod.c  super.o        xattr.c    xattr_user.c
```

  - ✓ insmod os_ext2.ko
  - ✓ lsmod | grep os_ext2 (잘 적재되었는지 확인)
  - ✓ cd ..

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2# insmod os_ext2.ko
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2# lsmod | grep os_ext2
os_ext2                73728  0
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3/os_ext2# cd ..
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# ls
append.c   Makefile  modules.order   os_ext2    ramdisk.ko   ramdisk.mod.c  ramdisk.o
create.sh  mnt       Module.symvers  ramdisk.c  ramdisk.mod  ramdisk.mod.o
```

2. File system Modification

- EXT2 Filesystem Mount, lookup 시 자신의 이름 출력
  - ✓ ext2로 포멧 후 os_ext로 마운트
  - ✓ mkfs.ext2 /dev/ramdisk
  - ✓ mount –t os_ext2 /dev/ramdisk ./mnt

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# mkfs.ext2 /dev/ramdisk
mke2fs 1.44.1 (24-Mar-2018)
Creating filesystem with 262144 4k blocks and 65536 inodes
Filesystem UUID: 820655ee-13bb-4475-8b87-1c951acaff33
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# mount -t os_ext2 /dev/ramdisk ./mnt
```

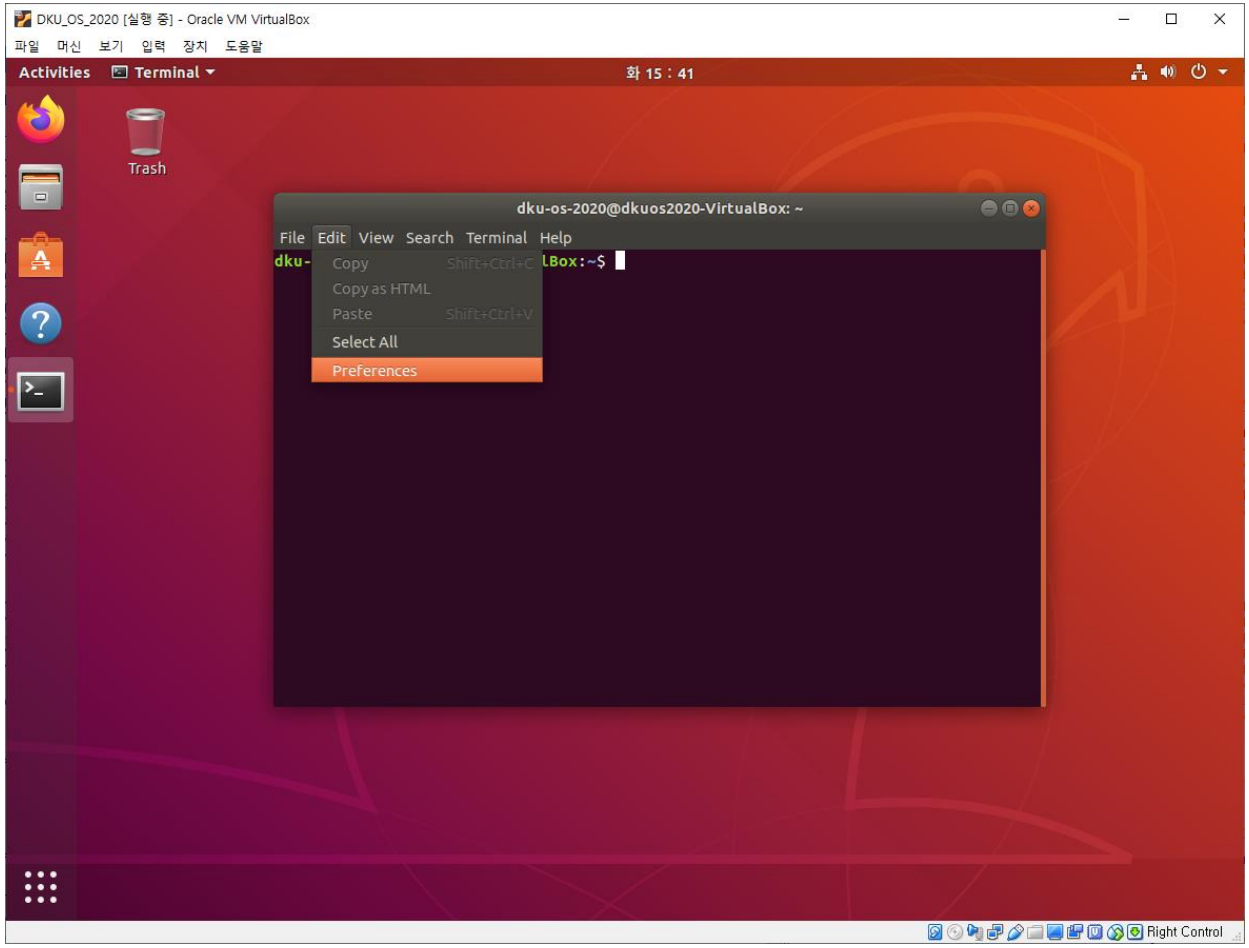
  - ✓ dmesg | grep os_ext2

```
root@ESL-LeeJY:/home/sys32153550/workspace/2020_1/OS_Lab3# dmesg | grep os_ext2
[2510165.993926] os_ext2 : Lee Jeyeon OS Lab3
```

  - ✓ 위와 같이 os_ext2 : 이름 OS Lab3 문구가 출력되었다면 성공

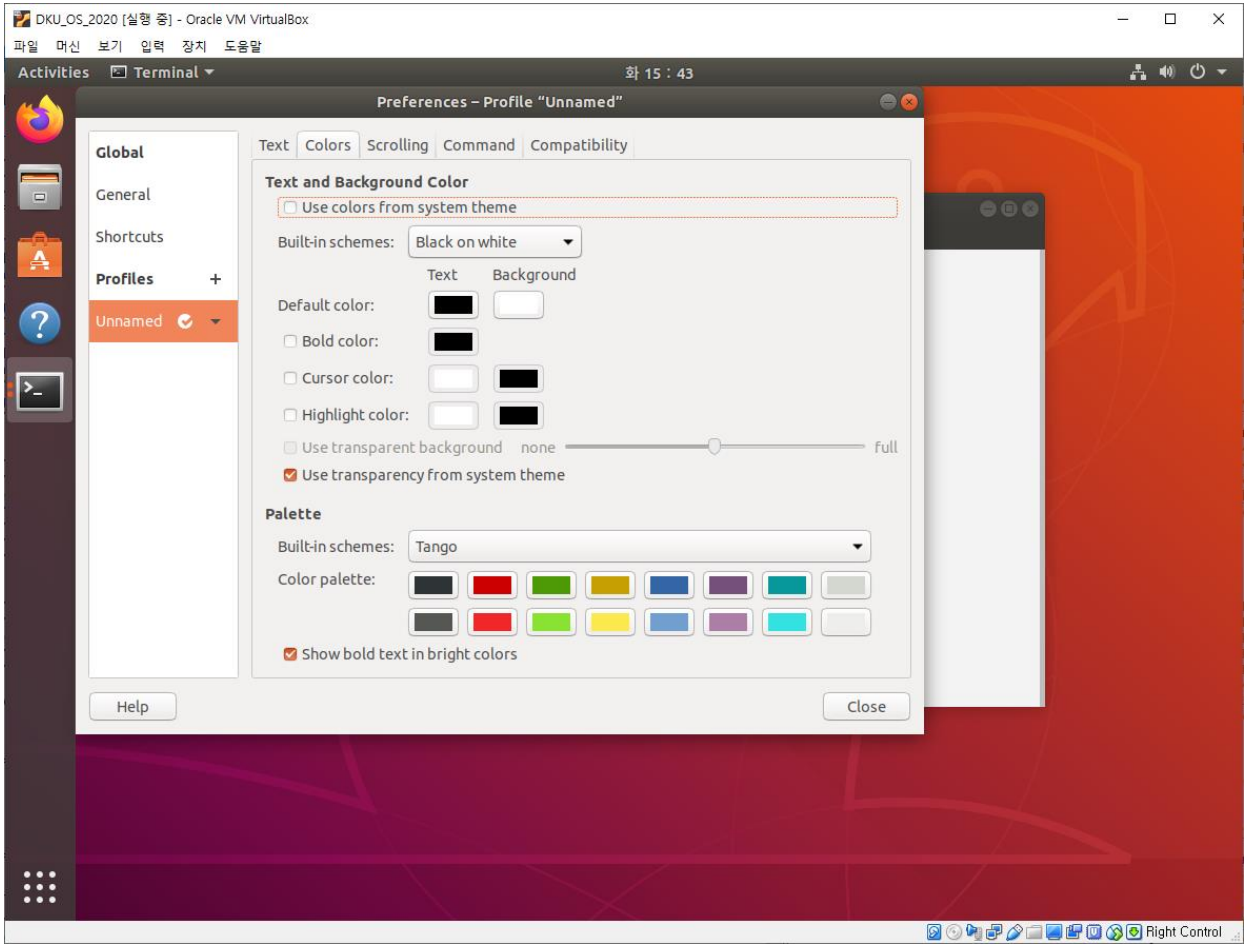
Appendix1

- 터미널 색상 변경하기
  - ✓ 과제 제출시 터미널 색상은 반드시 하얀색 배경으로 해야함
  - ✓ 터미널 왼쪽 위 메뉴에서 Edit -> Preferences



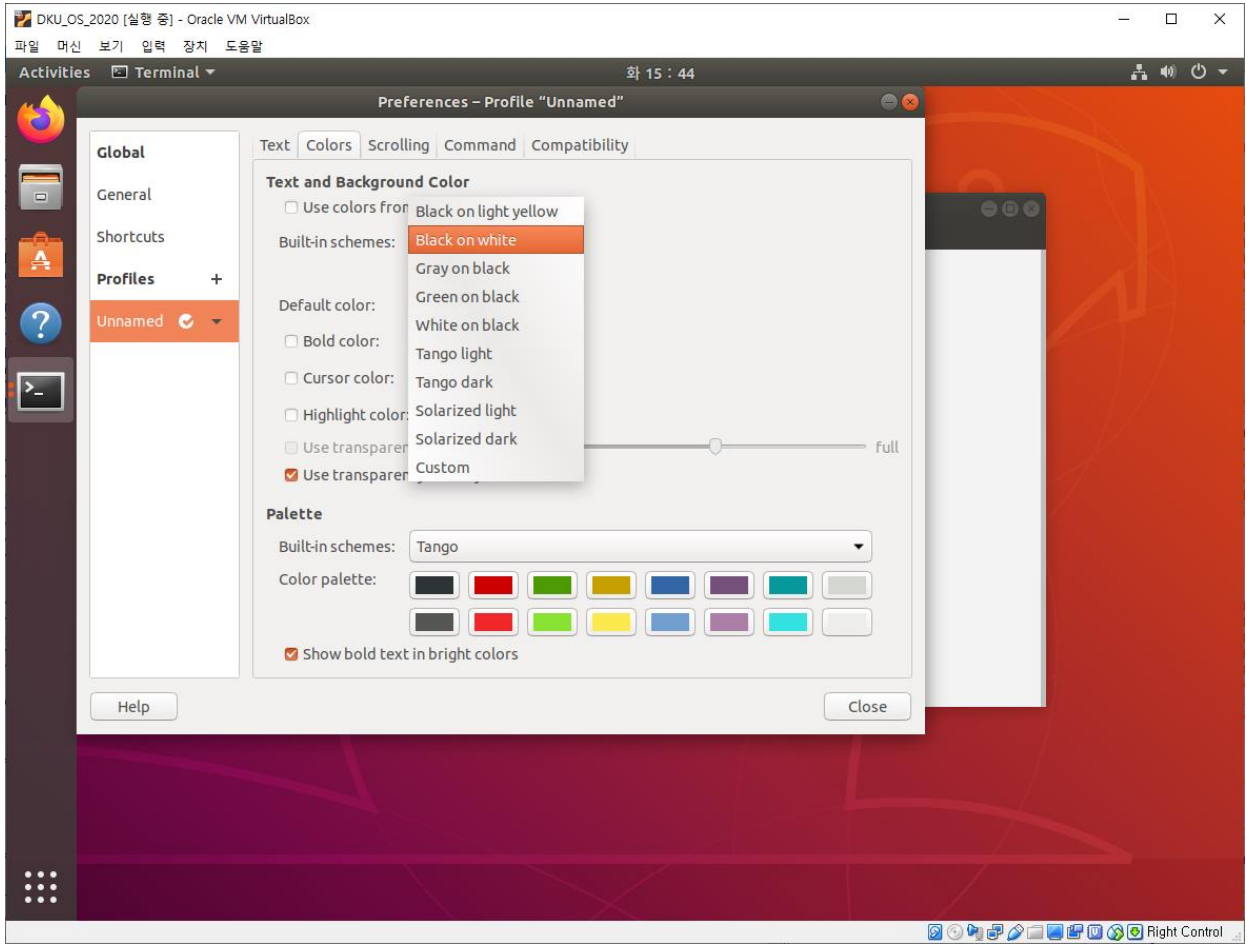
Appendix1

- 터미널 색상 변경하기
  - ✓ 과제 제출시 터미널 색상은 반드시 하얀색 배경으로 해야함
  - ✓ 위쪽 메뉴에서 Colors 선택후 Use colors from system theme 체크 해제



Appendix1

- 터미널 색상 변경하기
  - ✓ 과제 제출시 터미널 색상은 반드시 하얀색 배경으로 해야함
  - ✓ Built-in-schemes에서 Black on white 또는 Tango light 사용



- Q & A

Thank you!