EXT2 文件文件定位过程模拟实验(无理论版)

版权：

作者信息：

Alin Fang(Fang Yunlin)

MSN: cst05001@hotmail.com

G Talk: cst05001@gmail.com

blog: http://www.alinblog.cn

修改日期：

6 Aug, 2008

实验的目的：

做这次笔记的目的

1.是为了防忘

2.是为了和大家分享

理论：

硬盘就像一个大房间。数据就像是一坨书。

怎样把这些书摆放到房间里面，是你的自由。

但是当你要找你要的书，要找多久，你后果自负(-__-)

你要用多少时间找到你要的书，很大因素上取决于你摆放书的方法吧。至少多数人应该是这么认为的。

比如，我可以有这么几种摆书的方法：

- 把书一本本整齐的堆放在房子里
- 买个柜子，把书本分类放到柜子里
- 买几个柜子堆书，再做一个图书馆那样的索引卡，放在房间里，如果你需要哪方面的书，可以先通过索引卡确定书大概放在哪个位置

硬盘，好比就是这个堆书的房间。

文件系统，就是你管理这些书的方式。

下面解释一下一些概念：

block(块):

ext2 文件系统数据区域存储数据的最小单位。

在创建文件系统时指定。

我这里一个 block 有 1024byte

inode:

每个 inode 有 0x80 个 byte

里面记录的信息包括：

文件在分区里面的位置(以 block 为单位)。

记录这个位置在 inode 的第 41 – 44byte。

Inode 结构图图下：

Inode

2	1	2	File type and access rights
2	3	4	Owner identification
4	5	8	File length in bytes
4	9	12	Time of last file access
4	13	16	Time that inode last changed
4	17	20	Time that file contents last changed
4	21	24	Time of file deletion
2	25	26	Group identifier
2	27	28	Hard links counter
4	29	32	Number of data blocks of the file
4	33	36	File flags
4	37	40	Specific operating system information
4	41	44	Pointer to first data block
56	45	100	14 more pointers to data blocks
4	101	104	File version (for NFS)
4	105	108	File access control list
4	109	112	Directory access control list
4	113	116	Fragment address
8	117	124	Specific operating system information

此图引用自：http://homepage.smc.edu/morgan_david/cs40/analyze-ext2.htm

directory(目录)：

目录是一种特殊的文件，里面记录了该目录下的文件的 inode、类型、文件名长度、文件名

普通文件记录的是普通数据。

一些定义：

分区的最顶级目录的 inode 号是 2

$0x80_{(16)} == 128_{(10)}$

$0x400_{(16)} == 1024_{(10)}$

$0x800_{(16)} == 2048_{(10)}$

Group Descriptors: 组描述符。具体内容见下表：

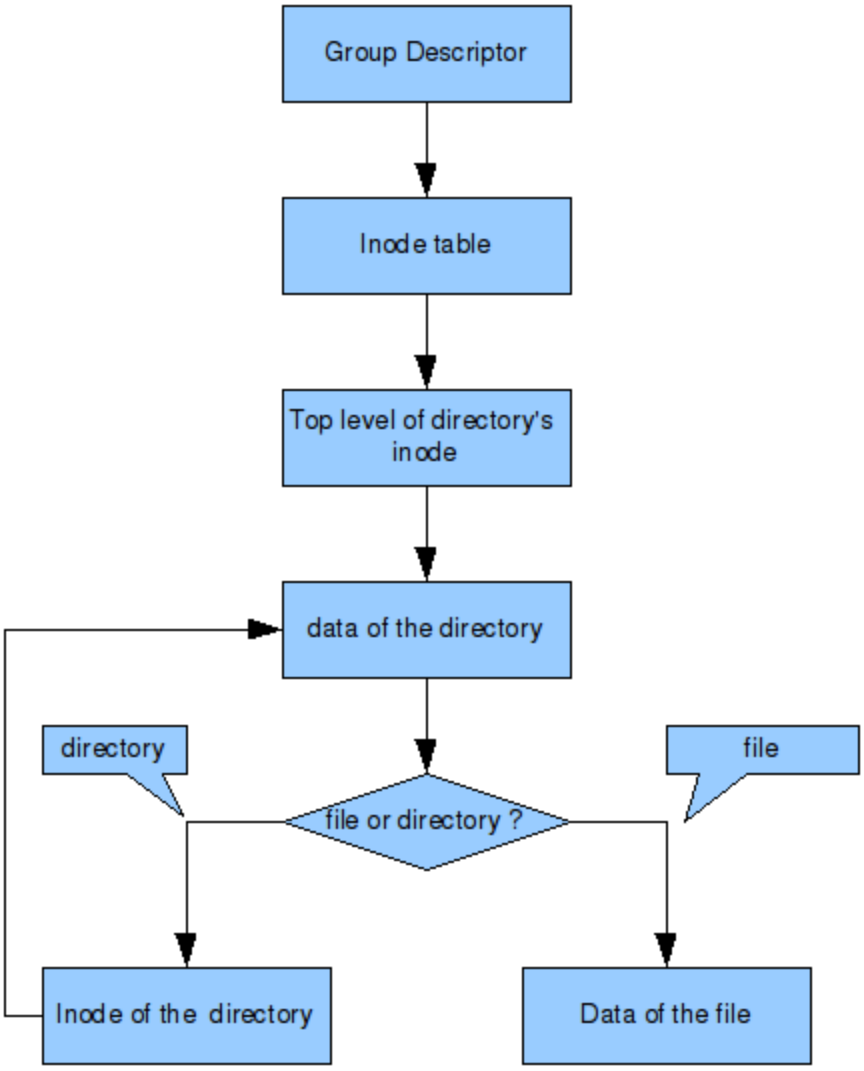
4	1	4	Block number of block bitmap
4	5	8	Block number of inode bitmap
4	9	12	Block number of first inode table block
2	13	14	Number of free blocks in the group
2	15	16	Number of free inodes in the group
2	17	18	Number of directories in the group
2	19	20	Alignment to word
4	21	24	Nulls to pad out 24 bytes

注：这个表格抄袭自 quner 的实验笔记。

Group Descriptors 的第 9-12 个字节记录了第一个 inode 表的在文件系统上的绝对位置(以 block 为单位)

文件定位流程：

找到 Group Descriptors，获取 inode table 的位置 => 找到 inode table 位置，确定最顶级目录 inode，确定该目录数据区的位置 => 找到顶级目录，确定要定位的的。



实验步骤：

虚拟出一个文件系统：

root@alin:/root# dd if=/dev/zero of=disk.img bs=1M count=1

1+0 records in

1+0 records out

1048576 bytes (1.0 MB) copied, 0.00320471 s, 327 MB/s

root@alin:/root# mkfs.ext2 disk.img

mke2fs 1.40.8 (13-Mar-2008)

disk.img is not a block special device.

Proceed anyway? (y,n) y

Filesystem label=

OS type: Linux

Block size=1024 (log=0)

Fragment size=1024 (log=0)

128 inodes, 1024 blocks

51 blocks (4.98%) reserved for the super user

First data block=1

Maximum filesystem blocks=1048576

1 block group

8192 blocks per group, 8192 fragments per group

128 inodes per group

Writing inode tables: done

Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 24 mounts or

180 days, whichever comes first. Use tune2fs -c or -i to override.

root@alin:/root# mkdir mnt

root@alin:/root# mount -o loop disk.img mnt/

root@alin:/root# mkdir mnt/dir1/

root@alin:/root# echo FangYunlin > mnt/dir1/name

```
root@alin:/root# echo 15011424628 > mnt/phone
root@alin:/root# umount mnt/
root@alin:/root#
```

让我们看下这个文件系统的一些属性：

```
root@alin:/root# tune2fs -l disk.img
tune2fs 1.40.8 (13-Mar-2008)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          f9211136-1617-4fe1-9336-ae637a1bc057
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      ext_attr resize_inode dir_index filetype sparse_super
Filesystem flags:         signed_directory_hash
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              128
Block count:              1024
Reserved block count:     51
Free blocks:              983
Free inodes:              114
First block:              1
Block size:               1024
Fragment size:            1024
Reserved GDT blocks:      3
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         128
Inode blocks per group:   16
Filesystem created:       Sat Sep  6 14:18:15 2008
Last mount time:          Sat Sep  6 14:19:21 2008
```

Last write time: Sat Sep 6 14:20:35 2008

Mount count: 1

Maximum mount count: 24

Last checked: Sat Sep 6 14:18:15 2008

Check interval: 15552000 (6 months)

Next check after: Thu Mar 5 14:18:15 2009

Reserved blocks uid: 0 (user root)

Reserved blocks gid: 0 (group root)

First inode: 11

Inode size: 128

Default directory hash: tea

Directory Hash Seed: 8847abf9-9ae7-4eeb-bfb9-92df433c42a7

root@alin:/root#

root@alin:/root#

我们发现这个 ext2 文件系统的 Block size 是 $1024_{(10)}$byte。即 $0x400_{(16)}$

重新梳理一下：

一个 inode 大小是 0x80

一个 block size 大小是 0x400

OK，那么开始模拟文件定位。

我们用 16 进制编辑器打开我们做出来的带有 ext2 文件系统的镜像。

这里推荐三个 16 进制编辑器：

khexedit

ghex

hexdump

我自己使用 khexedit。

root@alin:/root# khexedit disk.img &

[1] 21238

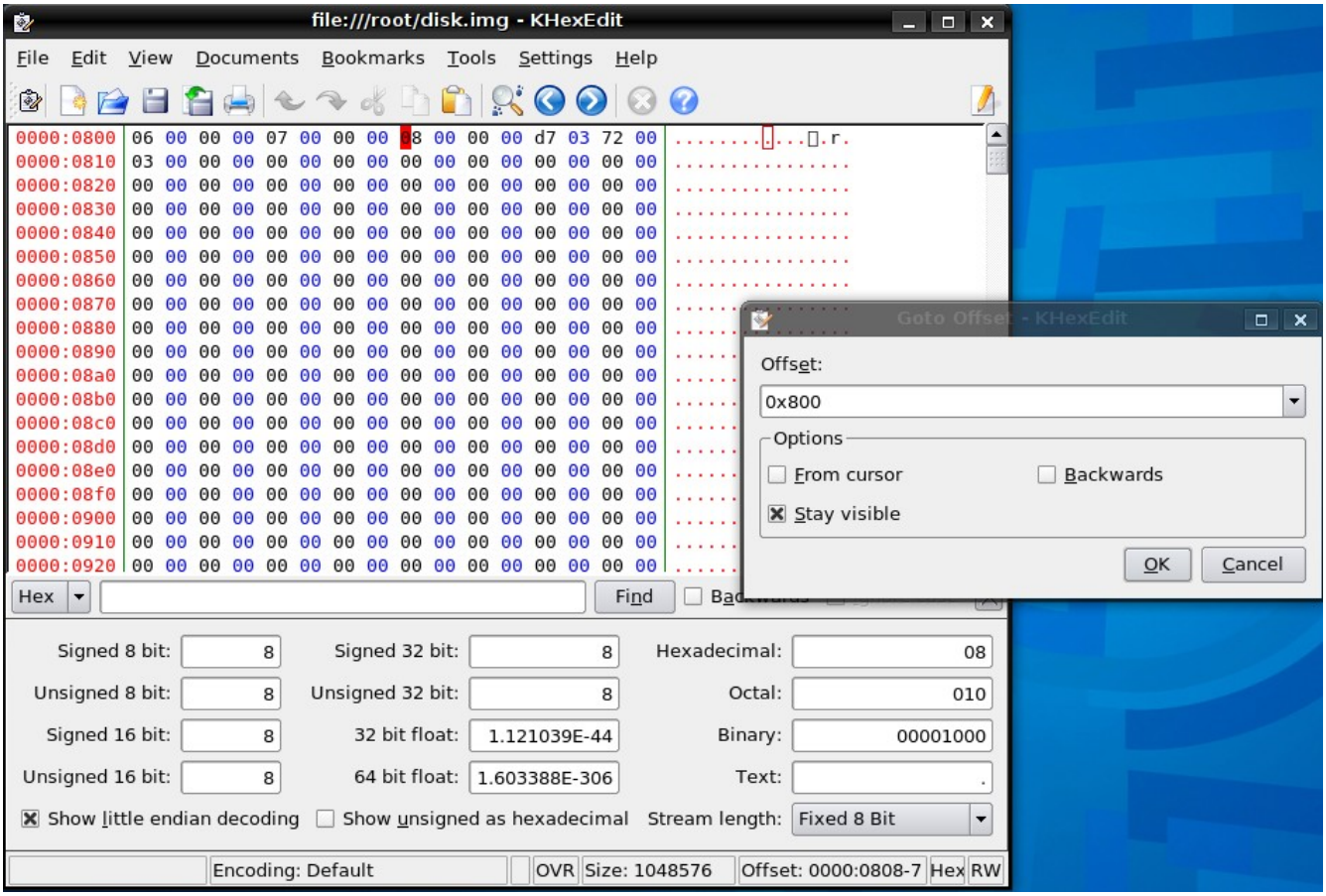
root@alin:/root#

我们跳到 0x800 位置，即 Group descriptor 的位置。其中第 9 – 12byte 记录了第一个 inode table 的位置。

我这里是 0x08。

所以第一个 inode 的位置在第 0x8 个 block 的位置。

0x08 X 0x400 = 0x2000

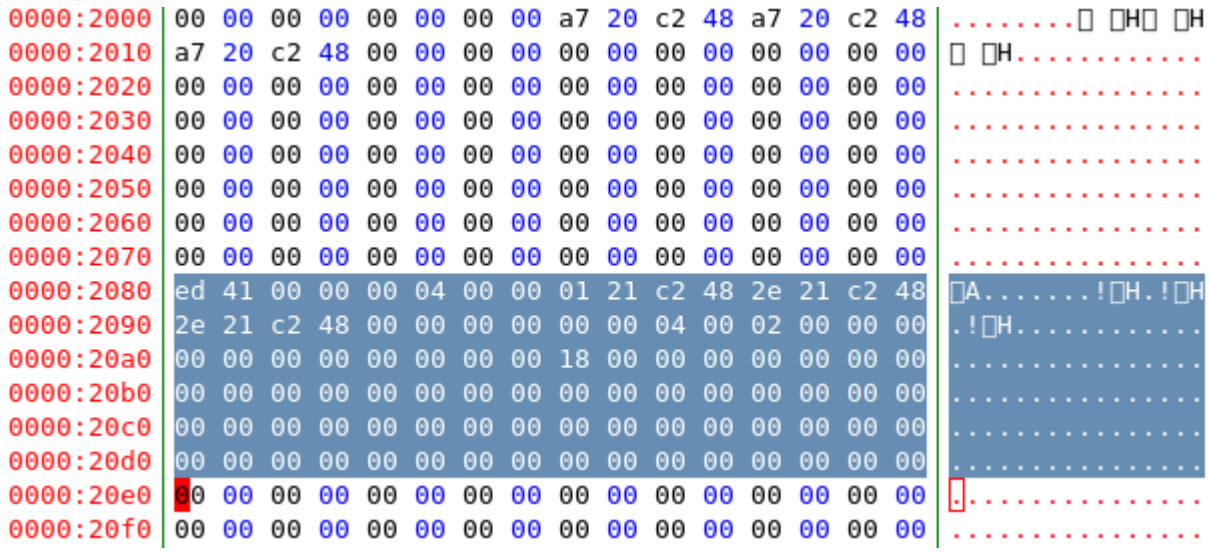


那么我们追踪到 inode table 的位置 0x2000:

我们已经知道：

- 一个文件系统的顶级目录的 inode 号是 2
- 一个 inode 的大小是 0x80byte。

所以可以知道 0x2080 到 0x20df 就是顶级目录

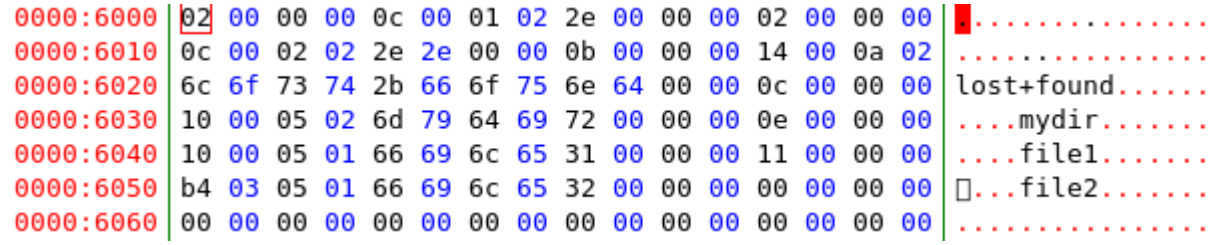


inode 的首块数据块指针存放于 41－44 的位置。从上图可以看出，这个分区的数据(顶级目录)存在于第 0x18 个 block。OK，我们去 0x18 个 block 看看。

重新提醒下，这里一个 block 的大小是 0x400byte。

所以 0x18 X 0x400 = 0x6000

下图是 0x6000 处的数据：



看到了点什么似曾相识的东西了吧？

没错！我们说过，目录也是一种文件。这个目录文件里面都存储了什么数据？我们可以看到文件以及下一级的文件夹！

目录下存放的文件的数据结构是这样的：

size	start	end	Directory Entry
4	1	4	Inode number
2	5	6	This directory entry's length
1	7	7	File name length
1	8	8	File type (1=regular file 2=directory)
	9?		File name

注：此图引用自

http://homepage.smc.edu/morgan_david/cs40/analyze-ext2.htm

(插一句：知道为什么 ext2/3 文件系统的文件名长度不能超过 256 个字符了吧？因为文件结构里面只能用 8bit 的数字来描述文件名长度， $2^8 = 256$)

那么我们打算定位 mydir 这个文件夹，查看里面的内容。

OK，根据这个表格，我们可以知道在文件名前面前 8bit 到前 5bit 记录了当前目录下该文件/文件夹的 inode 偏移量！

偏移量是指什么呢？就是指相对于 inode table 起始位置（这里是 0x2000）的相对偏移量。

另外，偏移量是以一个 inode(0x80bit)为单位的。

这里获得 mydir 文件夹的 inode 的偏移量是 0x0c。

所以 mydir 在分区中的绝对位置是：

0x2000 + 0x80 X (0x0c – 0x01) = 0x2580

之所以减去 0x01，是因为 inode 是从 0x00 开始计数，而不是 0x01。

那么我们去 0x2580 看看这里是什么东西：

```
0000:2580 ed 41 00 00 00 04 00 00 0b c5 c0 48 0a c5 c0 48 □A.......□□H.□□H
0000:2590 0a c5 c0 48 00 00 00 00 00 00 03 00 02 00 00 00 .□□H............
0000:25a0 00 00 00 00 00 00 00 00 26 00 00 00 00 00 00 00 ........&.......
0000:25b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:25c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

很眼花吧@_@

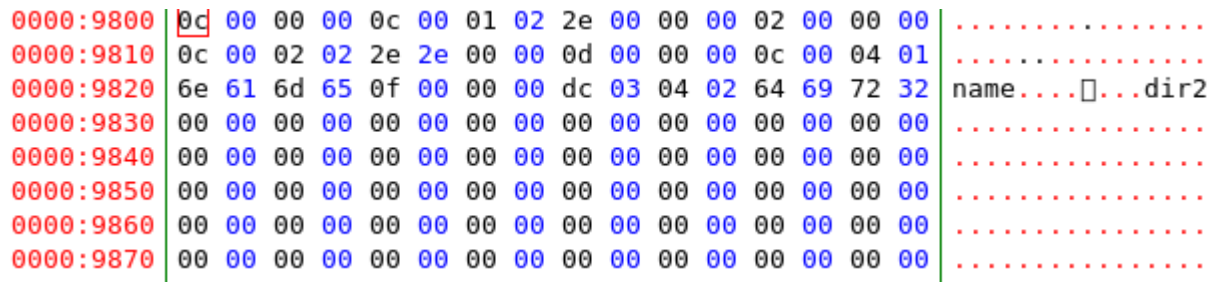
刚才我们说过了 inode 的结构。一个 inode 的数据块是在第 40 – 44byte 的位置。

这里是多少？

0x26!

那么我们去找 mydir 这个 inode 的数据。

0x26 X 0x400 = 0x9800



是不是看到了我们刚才在 mydir 目录下添加的东西了？哈哈。

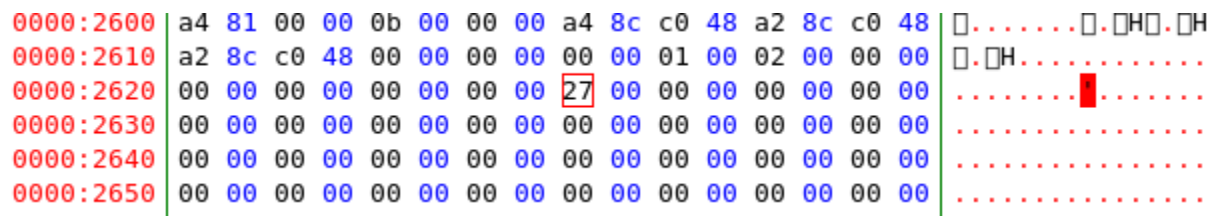
OK，我们再去定位一下 name 这个文件，查看下 name 里面的内容！

根据之前图示可以知道，一个目录文件里面的文件的 inode 是在文件名前 0x8 开始 4byte 的。

这里可以得出 name 的 inode 在 inode table 的偏移量是 0x0d。

OK，我们去看 name 文件的 inode！

0x2000 + 0x80 X (0x0d – 0x01) = 0x2600



根据 inode 结构，我们从 name 这个文件的 inode 得知 name 这个文件的数据块是在第 0x27 块。

我们到这里去看看！

0x400 X 0x27 = 0x9c00

哈哈！看到什么了？这个不是我们刚才创建的 name 文本文件的内容吗？！

原来 linux 就是这样找到我们看到的文件的啊！

后话：

这个实验笔记没有很多的理论。没有理论支撑的技术是脆弱的。从这个角度讲，这个实验笔记没有多少技术含量和营养。

希望如果有哥们看了这篇笔记，如果能引起深入学习 linux 的兴趣，那就很好了。

如果想深入研究，可以参考这个页面：

http://homepage.smc.edu/morgan_david/cs40/analyze-ext2.htm

另外……总觉得我的帖子在 CU 总是没人识货-__-lll这真的只是后话……

鸣谢：

wudx 同鞋，quner 同鞋。