[**理解ext4，第一部分(extents)**](http://blog.csdn.net/vah101/article/details/7426143)

EXT4作为EXT2、EXT3家族的下一代文件系统，已经鱼2008年10月被2.6.28版本的内核接纳。在本文撰写时，EXT4已经被许多发行版Linux作为默认的文件系统。EXT4文件系统在一定程度上向下兼容EXT2、EXT3，但是与原有的文件系统已经有很大的区别。类似Sleuthkit的文件系统工具软件，虽然有些功能在EXT4文件系统下还能工作，但它们还没有完全兼容EXT4的这些变化。

当我读了一些有关EXT4的报告，我对EXT4结构在磁盘上的实际分布情况、造成现有工具在EXT4文件系统上失效的原因产生了兴趣。所以我决定使用16进制编辑器来研究一下。这是此系列文章的第一篇。

**块地址**

EXT4使用了48位的块地址，我将会引用上面提到的文章来解释为何EXT4文件系统这样设计，为何这样会增大文件系统的大小。EXT4与EXT2、EXT3等传统Unix文件系统最大的区别在于使用了*extents*而不是间接块（inefficient indirect block）来标记文件内容。extent相似于NTFS文件系统中的运行(run)，本质上他们指示了组成extent的一系列文件块的起始地址、数量。一个文件可能由多段extent组成，但是EXT4尽可能保证文件连续存放。

这种新的块地址机制是造成绝大部分现存文件系统工具异常的原因。例如，当我在EXT4文件系统上创建一个新的文件，再用Sleuthkit工具包里的istat来查看，就会发现istat不能完全解析。

# **echo Here is a new file >testfile**

# **ls -li testfile**

918817 -rw-r--r-- 1 root root 19 2010-12-05 11:08 testfile

# **istat /dev/mapper/elk-home 918817**

inode: 918817

Allocated

Group: 112

Generation Id: 3173542730

uid / gid: 0 / 0

mode: rrw-r--r--

Flags:

size: 0

num of links: 1

Inode Times:

Accessed: Sun Dec 5 11:08:49 2010

File Modified: Sun Dec 5 11:08:49 2010

Inode Modified: Sun Dec 5 11:08:49 2010

Direct Blocks:

istat完全不能解析新的文件inode中的extent结构，所以没有能够显示出块地址。如果你仔细观察以上的输出，你也可以发现显示的文件大小为0字节，这是完全错误的。从另一个角度来看，inode元数据中许多其他的信息看起来是正确的，比如owner、group owner、MAC times等等。

实际上，EXT4开发者费了很大功夫来保证EXT4的inode能够最大限度地向下兼容EXT2、EXT3的inode结构。但是有些变化，比如extent、新的时间戳等，不能保证完全兼容。

**解析EXT4的inode**

我非常想通过16进制编辑器来观察EXT4的inode结构，但是这意味这需要精确的计算出inode在磁盘中的位置。幸运的是，EXT4文件系统上的超级块、块组描述表能够向下兼容，使用fsstat就能获得足够的信息：

# **fsstat /dev/mapper/elk-home**

FILE SYSTEM INFORMATION  
--------------------------------------------  
File System Type: Ext3  
[...]  
CONTENT INFORMATION  
--------------------------------------------  
Block Range: 0 - 113971199  
Block Size: 4096  
Free Blocks: 13506529

BLOCK GROUP INFORMATION  
--------------------------------------------  
Number of Block Groups: 3479  
Inodes per group: 8192  
Blocks per group: 32768  
[...]  
Group: 112:  
Inode Range: 917505 - 925696  
Block Range: 3670016 - 3702783  
Layout:  
Data bitmap: 3670016 - 3670016  
Inode bitmap: 3670032 - 3670032  
Inode Table: 3670048 - 3670559  
Data Blocks: 3670033 - 3670047, 3670560 - 3702783  
Free Inodes: 3281 (40%)  
Free Blocks: 0 (0%)  
Total Directories: 2  
[...]

我们可以通过之前的istat输出结果，来判断inode位于112块组。你也可以通过观察fsstat的输出中有关112块组的信息，并且可以看到编号为918817的inode正好在这个块组的范围内。

EXT4中最大的变化是它的inode是256个字节，而EXT2、EXT3文件系统中inode只有128个字节。这意味着每4K的块中会有16个inode，所以每个块组中会包含8192个inode，并占用起始位置的512个块。你可以看到当前这个块组中占用了512个块，从3670048到3670559。结果正是预期的。

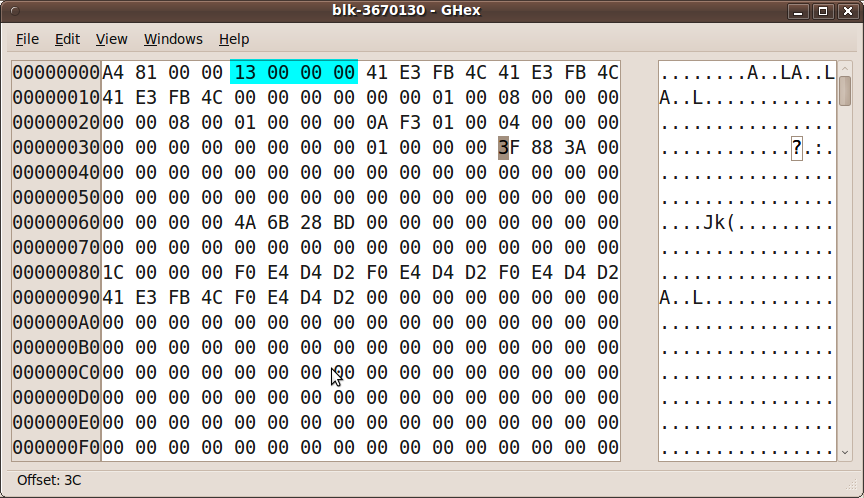
但是哪个块里包含了刚才创建文件的inode？在112组中，第一个inode的地址是917505. 用918817减去这个值，我们发现要找的inode从inode表起始位置算第1312个inode。很幸运，这个正好在一个块的开始，因为1312除以16正好得到82，即inode表中的第82个块。又因为在inode表中的第一个块地址为3670048，我们应该找到我们的inode在3670130块的第一个256字节中。

可以试用blkcat命令将这个块导出，这样就可以在16进制编辑器里分析

# **blkcat /dev/mapper/elk-home 3670130 >blk-3670130**

**EXT4的inode显微**

新的EXT4 inode是EXT3下文件系统的两倍，EXT4的开发者尝试尽可能不改变inode中前128位的试用方法。所以，你可以在第4-7字节中找到小端字节序的32位文件大小。

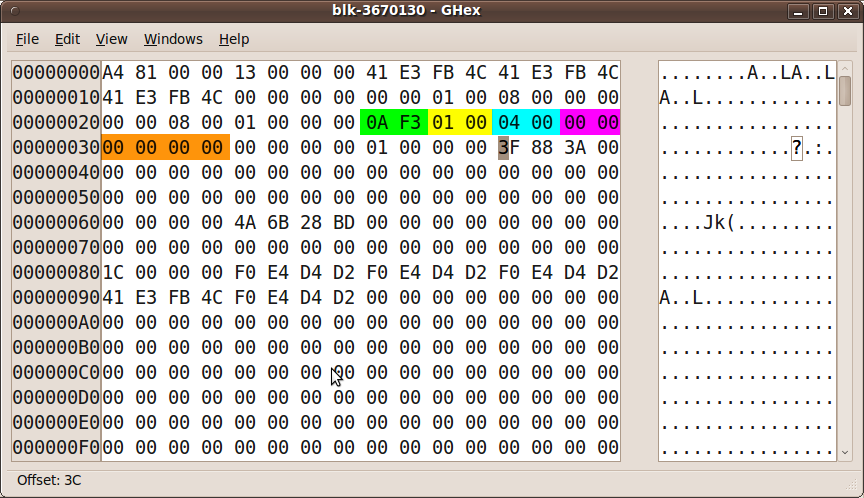


在小端字节序中，我们将它解析为文件长度为19字节，这正是我们生成文件的长度。看起来我们已经找到了正确的inode。

然而，因为EXT4试用了extent而不是块指向文件的内容，从第40到99这60个字节，用了保存extent信息而不再是块指针(block pointers)。extent结构是12个字节，所以你可以在每个inode中最多试用5个extent。然后，前12个字节是extent区(40到51字节)，被一个extent头结构占用，所以一个inode中实际上可以包含4个extent。

The values in the extent header are broken out as follows:

extent头中的值解析以后如下：



按照偏移字节数，将其细分，得到以下内容：

Bytes 40-41: Magic number (0xF30A = 62218)

42-43: Number of extents (0x0001 = 1)

44-45: Max number of extents (0x0004 = 4)

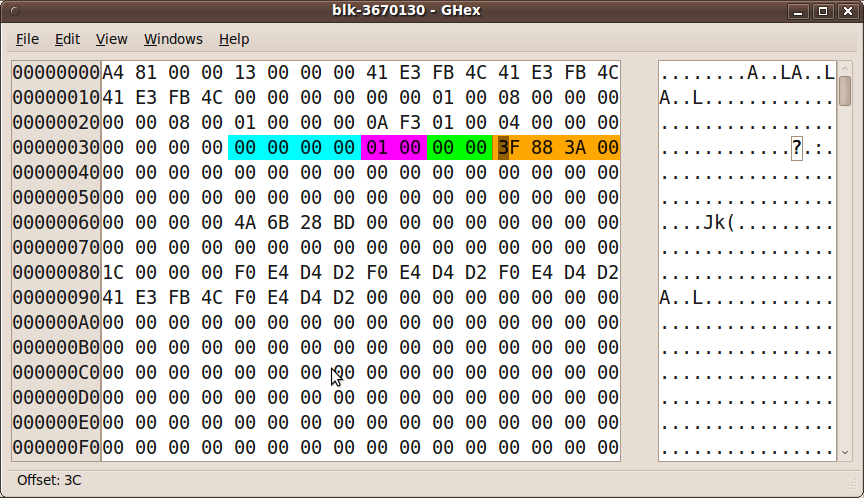
46-47: Depth of tree (0x0000 = 0)

48-51: Generation ID (0x00000000 = 0)

魔数（magic number）用来区分不同的extent实现方式。魔数的新增功能是，用来加强与旧的实现方式的兼容性。我们将在以后的章节中讨论“Depth of tree”和“Generation ID”值。

根据我们之前的讨论，一个inode中的extent数量，最大为4，并且在第44到45字节处予以标记。预计在未来，EXT4的实施者将会选出一种在inode保存附加extent的方式，所以预留两个字节的占位符给未来的方法。第42到43个字节处，告诉我们这个文件只有一个extent。

接下来的12个字节告诉我们extent中更多的信息：



Bytes 52-55: Logical block number (0x0000)

56-57: Number of blocks in extent (0x0001)

58-59: Upper 16 bits of physical block address (0x0000)

60-63: Lower 32 bits of physical block address (0x003A883F)

逻辑块号告诉我们，这个exten相对于文件t起始位置。在多extents的文件中，这将是非常重要的内容。但是，因为当前文件中，我们只有一个文件extent，它将其实有文件的开始，即逻辑块号为0。

接下来的两个字节告诉我们，这个extent中有几个块。因为是个小文件，所以我们只需要一个块。

下面6个字节告诉我们这个extent的第一个块的物理块号，即这个块在磁盘上的位置。现在的计算机系统期望值鱼16、32、64位对齐，48位多少有点问题。所以48位的块地址实际上代表了两个值：前两个字节给出了块地址的高16位，后4位给出了小端字节序的32位地址。在我们的例子里，我们将块号3835967解析位块地址0x0000003A883F。

下面尝试一下是否正确：

# blkcat /dev/mapper/elk-home 3835967

Here is a new file

因为没有更多的extent，inode中接下去的36个字节为空。这将是有趣的实验，看看是否这些未使用的字段可用于隐藏数据。