Righteous IT *Join the crusade!*

XFS (Part 4) – Block Directories

Posted on May 31, 2018 by Hal Pomeranz

In the previous installment (https://righteousit.wordpress.com/2018/05/25/xfs-part-3-short-form-directories/), we looked at small directories stored in “short form” in the inode. While these small directories can make up as much as 90% of the total directories in a typical Linux file system, eventually directories get big enough that they can no longer be packed into the inode data fork. When this happens, directory data moves out to blocks on disk.

In the inode (https://righteousit.wordpress.com/2018/05/23/xfs-part-2-inodes/), the data fork type (byte 5) changes to indicate that the data is no longer stored within the inode. Extents are used to track the location of the disk blocks containing the directory data. Here is the inode core and extent list for a directory that only occupies a single block:

```
0x000  49 4E 41 ED 03 02 00 00 00 00 00 00 00 00 00 00   INAí............
0x010  00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x020  5A EF 2F 87 03 1E 76 85 5A EF 2F 0F 32 59 22 1B   Zï/.. v.Zï/.2Y".
0x030  5A EF 2F 0F 32 59 22 1B 00 00 00 00 00 00 10 00   Zï/.2Y"........
0x040  00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 01   ................
0x050  00 00 23 01 00 00 00 00 00 00 00 00 3E 08 D0 05   ..#.........>.Ð.
0x060  FF FF FF FF 13 33 CB 38 00 00 00 00 00 00 00 01   ÿÿÿÿ.3Ë8........
0x070  00 00 00 20 00 00 62 6D 00 00 00 00 00 00 00 00   ... ..bm........
0x080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x090  5A EF 2F 0F 1B CF E4 64 00 00 00 00 04 08 E6 6D   Zï/..Ïäd......æm
0x0A0  E5 6C 3B 41 CA 03 4B 41 B1 5C DD 60 9C B7 DA 71   ål;AÊ.KA±\Ý`.·Úq
0x0B0  00 00 00 00 00 00 00 00 00 00 10 23 1C E0 00 01   ...........# à..
```

The data fork type is 2, indicating an extent list follows the inode core. Bytes 76-79 indicate that there is only a single extent. The extent starts at byte 176 (0x0B0), immediately after the inode core. The last 21 bits of the extent structure show that the extent only contains a single block. Parsing the rest of the extent yields a block address of 0x8118e7, or relative block 71911 in AG 2.

We can extract this block and examine it in our hex editor. Here is the data in the beginning of the block:

```
0x0000  58 44 42 33 AF 6A 41 6D 00 00 00 00 02 59 57 38   XDB3¯jAm.....YW8
0x0010  00 00 00 20 00 00 61 FE E5 6C 3B 41 CA 03 4B 41   ... ..aþål;AÊ.KA
0x0020  B1 5C DD 60 9C B7 DA 71 00 00 00 00 04 08 E6 6D   ±\Ý`.·Úq......æm
0x0030  05 10 09 48 00 00 00 00 00 00 00 00 00 00 00 00   .. H............
0x0040  00 00 00 00 04 08 E6 6D 01 2E 02 00 00 00 00 40   ......æm.......@
0x0050  00 00 00 00 04 15 9F A1 02 2E 2E 02 00 00 00 50   .......¡.......P
0x0060  00 00 00 00 04 17 97 6B 0C 30 31 5F 73 6D 61 6C   .......k 01_smal
0x0070  6C 66 69 6C 65 01 00 60 00 00 00 00 04 17 97 6C   lfile..`.......l
0x0080  0A 30 32 5F 62 69 67 66 69 6C 65 01 00 00 00 78    02_bigfile....x
0x0090  00 00 00 00 04 17 97 6D 0C 30 33 5F 73 6D 61 6C   .......m 03_smal
0x00A0  6C 66 69 6C 65 01 00 90 00 00 00 00 04 17 97 6E   lfile.........n
0x00B0  0A 30 34 5F 62 69 67 66 69 6C 65 01 00 00 00 A8    04_bigfile....¨
0x00C0  00 00 00 00 04 17 97 6F 0C 30 35 5F 73 6D 61 6C   .......o 05_smal
0x00D0  6C 66 69 6C 65 01 00 C0 00 00 00 00 04 17 97 70   lfile..À.......p
0x00E0  0A 30 36 5F 62 69 67 66 69 6C 65 01 00 00 00 D8    06_bigfile....Ø
0x00F0  00 00 00 00 04 17 97 71 0C 30 37 5F 73 6D 61 6C   .......q 07_smal
0x0100  6C 66 69 6C 65 01 00 F0 00 00 00 00 04 17 97 72   lfile..ð.......r
0x0110  0A 30 38 5F 62 69 67 66 69 6C 65 01 00 00 01 08    08_bigfile.....
0x0120  00 00 00 00 04 17 97 73 0C 30 39 5F 73 6D 61 6C   .......s 09_smal
0x0130  6C 66 69 6C 65 01 01 20 00 00 00 00 04 17 97 74   lfile.. .......t
0x0140  0A 31 30 5F 62 69 67 66 69 6C 65 01 00 00 01 38    10_bigfile....8
0x0150  00 00 00 00 04 17 97 75 0C 31 31 5F 73 6D 61 6C   .......u 11_smal
0x0160  6C 66 69 6C 65 01 01 50 00 00 00 00 04 17 97 76   lfile..P.......v
0x0170  0A 31 32 5F 62 69 67 66 69 6C 65 01 00 00 01 68    12_bigfile....h
0x0180  00 00 00 00 04 17 97 77 0C 31 33 5F 73 6D 61 6C   .......w 13_smal
0x0190  6C 66 69 6C 65 01 01 80 00 00 00 00 04 17 97 78   lfile.........x
0x01A0  0A 31 34 5F 62 69 67 66 69 6C 65 01 00 00 01 98    14_bigfile.....
```

The directory block begins with a 48 byte header:

```
0-3       Magic number                    XDB3
4-7       CRC32 checksum                  0xaf6a416d
8-15      Sector offset of this block     39409464

16-23     Last LSN update                 0x20000061fe
24-39     UUID                            e56c3b41-...-dd609cb7da71
40-47     inode that points to this block 0x0408e66d
```

You may compare the UUID and inode values in the directory block header with the corresponding values in the inode to see that they match.

The XFS documentation describes the sector offset field as the "block number". However, using the formula from Part 1 (https://righteousit.wordpress.com/2018/05/21/xfs-part-1-superblock/) of this series, we can calculate the physical block number of this block as:

```
(AG number) * (blocks per AG) + (relative block offset)
     2      *     2427136     +         71911   =   4926183
```

Multiply the block offset 4926183 by 8 sectors per block to get the sector offset value 39409464 that we see in the directory block header.

Following the header is a “free space” array that consumes 12 bytes, plus 4 bytes of padding to preserve 64-bit alignment. The free space array contains three elements which indicate where the three largest chunks of unused space are located in this directory block. Each element is a 2 byte offset and a 2 byte length field. The elements of the array are sorted in descending order by the length of each chunk.

In this directory block, there is only a single chunk of free space, starting at offset 1296 (0x0510) and having 2376 bytes (0x0948) of space. The other elements of the free space array are zeroed, indicating no other free space is available.

The directory entries start at byte 64 (0x040) and can be read sequentially like a typical Unix directory. However, XFS uses a hash-based lookup table, growing up from the bottom of the directory block, for more efficient searching:

```
0x0E50 00 00 00 00 00 00 05 10 00 00 00 2E 00 00 00 08  ................
0x0E60 00 00 17 2E 00 00 00 0A 3F 07 7C 6C 00 00 00 90  ....... ?.|l....
0x0E70 3F 07 7C EC 00 00 00 96 3F 07 7D 6C 00 00 00 84  ?.|ì....?.}l....
0x0E80 3F 07 7D EC 00 00 00 8A 3F 07 7F 6C 00 00 00 9C  ?.}ì....?..l....
0x0E90 3F 07 9C 6C 00 00 00 72 3F 07 9C EC 00 00 00 78  ?..l...r?..ì...x
0x0EA0 3F 07 9D 6C 00 00 00 66 3F 07 9D EC 00 00 00 6C  ?..l...f?..ì...l
0x0EB0 3F 07 9F 6C 00 00 00 7E 3F 07 BC 6C 00 00 00 54  ?..l...~?.¼l...T
0x0EC0 3F 07 BC EC 00 00 00 5A 3F 07 BD 6C 00 00 00 48  ?.¼ì...Z?.½l...H
0x0ED0 3F 07 BD EC 00 00 00 4E 3F 07 BF 6C 00 00 00 60  ?.½ì...N?.¿l...`
0x0EE0 3F 07 DC 6C 00 00 00 36 3F 07 DC EC 00 00 00 3C  ?.Ül...6?.Üì...<
0x0EF0 3F 07 DD 6C 00 00 00 2A 3F 07 DD EC 00 00 00 30  ?.Ýl...*?.Ýì...0
0x0F00 3F 07 DF 6C 00 00 00 42 3F 07 FC 6C 00 00 00 18  ?.ßl...B?.ül....
0x0F10 3F 07 FC EC 00 00 00 1E 3F 07 FD 6C 00 00 00 0C  ?.üì... ?.ýl...
0x0F20 3F 07 FD EC 00 00 00 12 3F 07 FF 6C 00 00 00 24  ?.ýì....?.ÿl...$
0x0F30 44 65 FD 31 00 00 00 99 44 65 FD 32 00 00 00 5D  Deý1....Deý2...]
0x0F40 44 65 FD 33 00 00 00 21 48 65 FD 31 00 00 00 8D  Deý3...!Heý1....
0x0F50 48 65 FD 32 00 00 00 51 48 65 FD 33 00 00 00 15  Heý2...QHeý3....
0x0F60 4A 65 FD 31 00 00 00 93 4A 65 FD 32 00 00 00 57  Jeý1....Jeý2...W
0x0F70 4A 65 FD 33 00 00 00 1B 4C 65 FD 31 00 00 00 81  Jeý3....Leý1....
0x0F80 4C 65 FD 32 00 00 00 45 4E 65 FD 31 00 00 00 87  Leý2...ENeý1....
0x0F90 4E 65 FD 32 00 00 00 4B 4E 65 FD 33 00 00 00 0F  Neý2...KNeý3....
0x0FA0 C4 65 FD 32 00 00 00 7B C4 65 FD 33 00 00 00 3F  Äeý2...{Äeý3...?
0x0FB0 C8 65 FD 32 00 00 00 6F C8 65 FD 33 00 00 00 33  Èeý2...oÈeý3...3
0x0FC0 CA 65 FD 32 00 00 00 75 CA 65 FD 33 00 00 00 39  Êeý2...uÊeý3...9
0x0FD0 CC 65 FD 31 00 00 00 9F CC 65 FD 32 00 00 00 63  Ìeý1....Ìeý2...c
0x0FE0 CC 65 FD 33 00 00 00 27 CE 65 FD 32 00 00 00 69  Ìeý3...'Îeý2...i
0x0FF0 CE 65 FD 33 00 00 00 2D 00 00 00 34 00 00 00 00  Îeý3...-...4....
```

The last 8 bytes of the directory block are a “tail record” containing two 4 byte values: the number of directory entries (0x34 or 52) and the number of unused entries (zero). Immediately preceding the tail record will be an array of 8 byte records, one record per directory entry (52 records in this case). Each record contains a hash value computed from the file name, and the offset in the directory block where the directory entry for that file is located. The array is sorted by hash value so that binary search can quickly find the desired record. The offsets are in 8 byte units.

The xfs_db program can compute hash values for us:

```
xfs_db> hash 03_smallfile
0x3f07fdec
```

If we locate this hash value in the array, we see the byte offset value is 0x12 or 18. Since the offset units are 8 bytes, this translates to byte offset 144 (0x090) from the start of the directory block.

Here are the first six directory entries from this block, including the entry for “03_smallfile”:

```
0x0040 00 00 00 00 04 08 E6 6D 01 2E 02 00 00 00 00 40  ......æm.......@
0x0050 00 00 00 00 04 15 9F A1 02 2E 2E 02 00 00 00 50  .......¡.......P
0x0060 00 00 00 00 04 17 97 6B 0C 30 31 5F 73 6D 61 6C  .......k 01_smal
0x0070 6C 66 69 6C 65 01 00 60 00 00 00 00 04 17 97 6C  lfile..`.......l
0x0080 0A 30 32 5F 62 69 67 66 69 6C 65 01 00 00 00 78   02_bigfile....x
0x0090 00 00 00 00 04 17 97 6D 0C 30 33 5F 73 6D 61 6C  .......m 03_smal
0x00A0 6C 66 69 6C 65 01 00 90 00 00 00 00 04 17 97 6E  lfile..........n
0x00B0 0A 30 34 5F 62 69 67 66 69 6C 65 01 00 00 00 A8   04_bigfile....¨
```

Directory entries are variable length, but always 8 byte (64-bit) aligned. The fields in each directory entry are:

```
Len (bytes)       Field
===========       =====
     8            Inode number
     1            File name length
     varies       File name
     1            File type
     varies       Padding for alignment
     2            Byte offset of this directory entry
```

64-bit inode addresses are always used. This is different from “short form” directories, where 32-bit inode addresses will be used if possible.

File name length is a single byte, limiting file names to 255 characters. The file type byte uses the same numbering scheme we saw in “short form” directories:

```
1   Regular file
2   Directory
3   Character special device
4   Block special device
5   FIFO
6   Socket
7   Symlink
```

Padding for alignment is only included if necessary. Our “03_smallfile” entry starting at offset 0x090 is exactly 24 bytes long and needs no padding for alignment. You can clearly see the padding in the “.” and “..” entries starting at offset 0x040 and 0x050 respectively.

Deleting a File

If we remove “03_smallfile” from this directory, the inode updates similarly to what we saw with the “short form” directory in the last installment of this series. The mtime and ctime values are updated, and the CRC32 and Logfile Sequence Number fields as well. The file size does not change, since the directory still occupies one block.

The “tail record” and hash array at the end of the directory block change:

```
0x0E50 00 00 00 00 00 00 05 10 00 00 00 2E 00 00 00 08 ................
0x0E60 00 00 17 2E 00 00 00 0A 3F 07 7C 6C 00 00 00 90 ....... ?.|l....
0x0E70 3F 07 7C EC 00 00 00 96 3F 07 7D 6C 00 00 00 84 ?.|ì....?.}l....
0x0E80 3F 07 7D EC 00 00 00 8A 3F 07 7F 6C 00 00 00 9C ?.}ì....?..l....
0x0E90 3F 07 9C 6C 00 00 00 72 3F 07 9C EC 00 00 00 78 ?..l...r?..ì...x
0x0EA0 3F 07 9D 6C 00 00 00 66 3F 07 9D EC 00 00 00 6C ?..l...f?..ì...l
0x0EB0 3F 07 9F 6C 00 00 00 7E 3F 07 BC 6C 00 00 00 54 ?..l...~?.¼l...T
0x0EC0 3F 07 BC EC 00 00 00 5A 3F 07 BD 6C 00 00 00 48 ?.¼ì...Z?.½l...H
0x0ED0 3F 07 BD EC 00 00 00 4E 3F 07 BF 6C 00 00 00 60 ?.½ì...N?.¿l...`
0x0EE0 3F 07 DC 6C 00 00 00 36 3F 07 DC EC 00 00 00 3C ?.Ül...6?.Üì...<
0x0EF0 3F 07 DD 6C 00 00 00 2A 3F 07 DD EC 00 00 00 30 ?.Ýl...*?.Ýì...0
0x0F00 3F 07 DF 6C 00 00 00 42 3F 07 FC 6C 00 00 00 18 ?.ßl...B?.ül....
0x0F10 3F 07 FC EC 00 00 00 1E 3F 07 FD 6C 00 00 00 0C ?.üì... ?.ýl...
0x0F20 3F 07 FD EC 00 00 00 00 3F 07 FF 6C 00 00 00 24 ?.ýì....?.ÿl...$
0x0F30 44 65 FD 31 00 00 00 99 44 65 FD 32 00 00 00 5D Deý1....Deý2...]
0x0F40 44 65 FD 33 00 00 00 21 48 65 FD 31 00 00 00 8D Deý3...!Heý1....
0x0F50 48 65 FD 32 00 00 00 51 48 65 FD 33 00 00 00 15 Heý2...QHeý3....
0x0F60 4A 65 FD 31 00 00 00 93 4A 65 FD 32 00 00 00 57 Jeý1....Jeý2...W
0x0F70 4A 65 FD 33 00 00 00 1B 4C 65 FD 31 00 00 00 81 Jeý3....Leý1....
0x0F80 4C 65 FD 32 00 00 00 45 4E 65 FD 31 00 00 00 87 Leý2...ENeý1....
0x0F90 4E 65 FD 32 00 00 00 4B 4E 65 FD 33 00 00 00 0F Neý2...KNeý3....
0x0FA0 C4 65 FD 32 00 00 00 7B C4 65 FD 33 00 00 00 3F Äeý2...{Äeý3...?
0x0FB0 C8 65 FD 32 00 00 00 6F C8 65 FD 33 00 00 00 33 Èeý2...oÈeý3...3
0x0FC0 CA 65 FD 32 00 00 00 75 CA 65 FD 33 00 00 00 39 Êeý2...uÊeý3...9
0x0FD0 CC 65 FD 31 00 00 00 9F CC 65 FD 32 00 00 00 63 Ìeý1....Ìeý2...c
0x0FE0 CC 65 FD 33 00 00 00 27 CE 65 FD 32 00 00 00 69 Ìeý3...'Îeý2...i
0x0FF0 CE 65 FD 33 00 00 00 2D 00 00 00 34 00 00 00 01 Îeý3...-...4....
```

The tail record still shows 34 entries, but one of them is now unused. If we look at the entry for hash 0x3F07FDEC, we see the offset value has been zeroed, indicating an unused record.

We also see changes at the beginning of the block:

```
0x0000 58 44 42 33 DA 98 6C F3 00 00 00 00 02 59 57 38 XDB3Ú.ló.....YW8
0x0010 00 00 00 20 00 00 78 5F E5 6C 3B 41 CA 03 4B 41 ... ..x_ål;AÊ.KA
0x0020 B1 5C DD 60 9C B7 DA 71 00 00 00 00 04 08 E6 6D ±\Ý`.·Úq......æm
0x0030 05 10 09 48 00 90 00 18 00 00 00 00 00 00 00 00 .. H............
0x0040 00 00 00 00 04 08 E6 6D 01 2E 02 00 00 00 00 40 ......æm.......@
0x0050 00 00 00 00 04 15 9F A1 02 2E 2E 02 00 00 00 50 .......¡.......P
0x0060 00 00 00 00 04 17 97 6B 0C 30 31 5F 73 6D 61 6C .......k 01_smal
0x0070 6C 66 69 6C 65 01 00 60 00 00 00 00 04 17 97 6C lfile..`.......l
0x0080 0A 30 32 5F 62 69 67 66 69 6C 65 01 00 00 00 78  02_bigfile....x
0x0090 FF FF 00 18 04 17 97 6D 0C 30 33 5F 73 6D 61 6C ÿÿ.....m 03_smal
0x00A0 6C 66 69 6C 65 01 00 90 00 00 00 00 04 17 97 6E lfile..........n
0x00B0 0A 30 34 5F 62 69 67 66 69 6C 65 01 00 00 00 A8  04_bigfile....¨
```

The free space array now uses the second element, showing 24 (0x18) bytes free at byte offset 0x90– the location where the “03_smallfile” entry used to reside.

Looking at offset 0x90, we see that the first two bytes of the inode field are overwritten with 0xFFFF, indicating an unused entry. The next two bytes are the length of the free space. Again we see 0x18, or 24 bytes.

However, since inode addresses in this file system fit in 32 bits, the original inode address associated with this file is still clearly visible. The rest of the original directory entry is untouched until a new entry overwrites this space. This should make file recovery easier.

Not Quite Done With Directories

When directories get large enough to occupy multiple blocks, the directory structure gets more complicated. We'll examine larger directories in our next installment.

Posted in Forensics, UncategorizedTagged File Systems, Linux, XFS

Published by Hal Pomeranz

Independent Computer Forensics and Information Security consultant. Expert Witness. Trainer. *View all posts by Hal Pomeranz*