Профиль: Аноним (вход | регистрация) неRU opennet.me
OpenNET www.opennet.ru НОВОСТИ (+) КОНТЕНТ WIKI MAN'ы ФОРУМ Поиск (теги)

Структура Файловой Системы XFS

Николай (unDEFER) Кривченков

Содержание

Введение

Эта статья описывает структуру файловой системы XFS (disk-layout). Она написана «по горячим следам» после создания утилиты build_xfs для проекта anyfs-tools (*http://anyfs-tools.sourceforge.net*).

Причиной написания этой статьи стало практически полное отсутствие описания структуры XFS не только на русском, но даже на английском есть только старые документы о дизайне XFS (довольно устаревшие), а также `man xfs_db` (помог разобраться в некоторых сложных моментах, с подсказки самого Nathan Scott'a). Ну и разумеется - исходные коды, точнее заголовки (к сожалению описания структур там довольно запутаны) .

Все они довольно разрознены и такого описания как в 16-ой главе книги В.Костромина "Linux для пользователя":
http://www.linuxcenter.ru/lib/books/kostromin
по Ext2FS, по XFS не найти.

Эта статья не является полным описанием структуры XFS. Она описывает её лишь в той степени, которая мне была необходима для написания build_xfs. Таким образом останутся незатронутыми назначения real-time блоков, структура журнала, расширенные аттрибуты и многие другие поля в структурах, от понимания назначения, которых я сам всё ещё далёк.

Перед знакомством с этой статьёй желательно иметь некоторое представление о том как устроены Файловые Системы вообще и Ext2FS в частности.

Итак, передо мной лежат 9 листов рукописи структур данных XFS, а ниже я попробую привести это всё в хоть каком-то упорядоченном виде. Писать постараюсь по возможности живым языком, чтобы читать эту статью было не слишком скучно.

Сообщения об ошибках, поправки и пожелания по поводу этой статьи прошу направлять по адресу: *undefer@gmail.com*

Ссылки на эту статью

Постоянный адрес этой статьи на сайте LUGR:
http://www.lugr.ru/articles/XFS-layout

Сжатый html:
http://www.lugr.ru/files/XFS-layout-ru.html.gz

PDF (сжатый gzip):
http://www.lugr.ru/files/XFS-layout-ru.pdf.gz

Latex оригинал:
http://www.lugr.ru/files/XFS-layout-ru-latex.tar.gz

Общая структура XFS

Итак, вся файловая система делится на так называемые *Группы Выделения* (*Allocation Groups или просто AG*), аналог групп Блоков в Ext2FS.

Размер/количество и прочее описание Групп Выделения находится в суперблоке, а суперблок находится в начале каждой из Групп Выделения (т.е. здесь пока всё как и в Ext2FS), а потому сразу перейдём к описанию её структуры.

Структура Группы Выделения

По меньшей мере первые 0x800 байт (2048 байт, 2 Кб) каждой Группы Выделения имеют совершенно одинаковый формат. Причём нулевая (будем считать всё от нуля) Группа Выделения (а вместе с ней и нулевой суперблок) располагается прямо в начале устройства. Здесь, важное отличие от абсолютного числа других Файловых Систем: когда ребятами из SGI проектировалась их файловая система XFS ещё для платформы IRIX, они и думать не думали ни о каких загрузчиках в начале диска, а потому даже не пытайтесь ставить загрузчик в раздел с XFS.

Группа Выделения делится на ещё четыре структуры:

- Суперблок
- Информация о свободных блоках
- Информация о выделенных и свободных инф.узлах
- Блоки выделенные для расширения Би-деревьев

Суперблок

Итак, в начале Группы Выделения находится суперблок. Определение структуры *struct xfs_sb* его описывающую вы можете найти в файле /usr/include/xfs/xfs_sb.h.

Я бы мог привести цитату этого файла, как обычно это делается в таких случаях, но для описания структуры в данном случае это вовсе не удобно уже потому, что названия типов вроде *xfs_drfsbno_t* ничего не говорят об их размере.

А потому, я буду записывать структуры в формате, который был более удобен для меня. Надеюсь, он также будет удобен и для Вас.

Так я буду описывать структуры используя 5 полей:

- Смещение от начала структуры (в шестнадцатиричной системе счисления)
- Размер поля в битах
- Название элемента в структуре
- Некоторое типичное значение (которые я брал с ФС сразу после форматирования некоторого тестового образа)
- Комментарий

```
Структура xfs_sb
{

0x00        32     sb_magicnum     = "XFSB"     /* магическое число */
0x04        32     sb_blocksize    = 4096       /* размер блока */
0x08        64     sb_dblocks      = 131072     /* блоков данных */
0x10        64     sb_rblocks      = 0          /* realtime блоков */
0x18        64     sb_rextents     = 0          /* realtime фрагментов */
0x20       128     sb_uuid                      /* уникальный
                                                       идентификатор ФС */
0x30        64     sb_logstart     = 65540      /* начало внутреннего журнала
*/
0x38        64     sb_rootino      = 128        /* корневой инф. узел */
0x40        64     sb_rbmino       = 129        /* bitmap inode for
                                                        realtime extents */
```

```
  0x48        64     sb_rsumino      = 130        /* summary inode for
                                                     rt bitmap */
  0x50        32     sb_rextsize     = 16         /* realtime extent size,
                                                      blocks */
  0x54        32     sb_agblocks     = 16384      /* размер Группы Выделения */
  0x58        32     sb_agcount      = 8          /* число Групп Выделения */
  0x5C        32     sb_rbmblocks    = 0          /* number of rt bitmap blocks
  */
  0x60        32     sb_logblocks    = 1200       /* размер журнала в блоках */
  0x64        16     sb_versionnum   = 0x3084     /* header version ==
                                                        XFS_SB_VERSION */
  0x66        16     sb_sectsize     = 512        /* размер сектора устройства */
  0x68        16     sb_inodesize    = 256        /* размер инф.узла */
  0x6A        16     sb_inopblock    = 16         /* инф.узлов на блок */
  0x6C      8x12     sb_fname[12]    = "Белая_Метка" /* имя (метка) ФС */
  0x78         8     sb_blocklog     = 12         /* log_2(sb_blocksize) */
  0x79         8     sb_sectlog      = 9          /* log_2(sb_sectsize) */
  0x7A         8     sb_inodelog     = 8          /* log_2(sb_inodesize) */
  0x7B         8     sb_inopblog     = 4          /* log_2(sb_inopblock) */
  0x7C         8     sb_agblklog     = 14         /* log_2(sb_agblocks)
                                                     округлённый в сторону
                                                     увеличения */
  0x7D         8     sb_rextslog     = 0          /* log_2(sb_rextents) */
  0x7E         8     sb_inprogress   = 1          /* mkfs в прогрессе,
                                                       не монтировать */
  0x7F         8     sb_imax_pct     = 25         /* max % of fs for inode
                                                      space statistics */

  0x80        64     sb_icount       = 64         /* выделено инф. узлов */
  0x88        64     sb_ifree        = 61         /* свободно инф. узлов */
  0x90        64     sb_fdblocks     = 129840     /* свободно блоков данных */
  0x98        64     sb_frextents    = 0          /* free realtime extents */

  0xA0        64     sb_uquotino     = 0          /* user quota inode */
  0xA8        64     sb_gquotino     = 0          /* group quota inode */
  0xB0        16     sb_qflags       = 0          /* quota flags */
  0xB2         8     sb_flags        = 0          /* misc. flags */
  0xB3         8     sb_shared_vn    = 0          /* shared version number */
  0xB4        32     sb_inoalignmt   = 2          /* inode chunk alignment,
                                                       fsblocks */
  0xB8        32     sb_unit         = 0          /* stripe or raid unit */
  0xBC        32     sb_width        = 0          /* stripe or raid width */
  0xC0         8     sb_dirblklog    = 0          /* логарифм размера блока
                                                     директорий измеренного
                                                     в блоках ФС */
  0xC1         8     sb_logsectlog   = 0          /* ilog2 of the log
                                                      sector size */
  0xC2        16     sb_logsectsize  = 0          /* sector size for
                                                     the log, bytes */
  0xC4        32     sb_logsunit     = 0          /* stripe unit size
                                                      for the log */
  0xC8        32     sb_features2    = 0          /* additional feature bits */
  0xCC
  }
```

О нумерации инф.узлов

Вторым же полем в суперблоке идёт размер блока (*sb_blocksize*). И я думаю не надо объяснять с каким смещением в файловой системе должен находится скажем блок 12. Правильно, в нашем случае, со смещением 12 x 4096.

Файловая система XFS любопытна тем, что такой же метод как для блоков применён и в отношении инф.узлов, т.е. номер инф.узла одновременно однозначно определяет и его местоположение в файловой системе. В связи с чем существуют поля размера инф.узла (0x68 *sb_inodesize*) и числа инф.узлов на блок (0x6A *sb_inopblock*). Таким образом размер инф.узла должен быть меньше размера блока в число раз являющемся степенью двойки.

В приведённом примере корневой узел имеет номер 128 (0x38 *sb_rootino*). Таким образом его смещение можно вычислить как 128 x 256.

Ещё одной особенностью такой нумерации инф.узлов является то, что существование инф.узла 128, вовсе не означает существование инф.узлов 0-127. В данном примере выделены инф.узлы 128-191, а других инф.узлов пока не существует, но когда заполняться эти свободные 61 инф.узел, могут быть выделены ещё 64 инф.узла. Впрочем, об этом - ниже.

Ещё о нумерации блоков и инф.узлов

Посмотрим на размер Группы Выделения в приведённом примере. Поле 0x54 *sb_agblocks* равно 16384 блока.

Если, теперь, я попрошу Вас назвать смещение блока номер 16384, выходящем за пределы первой Группы Выделения, вы смело можете назвать 16384 x 4096 и.. не ошибётесь.

Однако, не спешите думать что всё так просто. Формула <смещение блока> = <номер блока> x <размер блока> верна не всегда. Обратите внимание на поле 0x7C *sb_agblklog*. Это логарифм размера Группы Выделения. В нашем случае 2^14 = 16384 и всё сходится. Но нам повезло потому, что размер образа (512 Мб) есть степень двойки. В реальности файловая система редко имеет такой красивый размер...

Пусть теперь у нас размер Группы Выделения равен 16000... Угадайте, какое смещение тогда имеет блок 16384? 16384 x 4096? А вот и нет - 16000 x 4096.

Дело всё в том, что поле *sb_agblklog* существует не просто так, оно действительно округляет размер Группы Выделения, и для Групп Выделения в 16000 блоков, *sb_agblklog* по-прежнему равно 14.

Таким образом номер блока равен:

```
(<номер Группы Выделения> << sb_agblklog) | <Номер блока в Группе Выделения>
```

Схематично:

```
Номер блока (52 бита):

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAANNNNNNNNNNNNNN
+-----------------------------------++------------+
|                                   |
+-- (52 - sb_agblklog) бит          +-- (sb_agblklog) бит
    номер Группы Выделения                Номер блока
                                      в Группе Выделения
```

Для номеров инф.узлов будет тоже самое с заменой *sb_agblklog* на *(sb_agblklog + sb_inopblog)*.

Информация о свободных блоках

Информация о Свободных блоках Группы Выделения располагается в ней со смещением **0x0200**.

Определение структуры *xfs_agf* находится в файле /usr/include/xfs/xfs_ag.h.

```
Структура xfs_agf
{
0x00        32     agf_magicnum    = "XAGF"     /* магическое число */
0x04        32     agf_versionnum  = 1          /* версия agf-заголовка */
0x08        32     agf_seqno       = 0          /* номер Группы Выделения */
0x0C        32     agf_length      = 16384      /* размер Группы Выделения */

0x10        32     agf_roots[XFS_BNO]   = 1     /* Корень Би-дерева свободных
                                                    фрагментов с сортировкой
                                                    по номеру */
0x14        32     agf_roots[XFS_CNUM]  = 2     /* Корень Би-дерева свободных
                                                    фрагментов с сортировкой
                                                    по размеру */
0x18        32     agf_spare0      = 0          /* запас */
0x1C        32     agf_levels[XFS_BNO]  = 1     /* Уровень Би-дерева свободных
                                                    фрагментов с сортировкой
                                                    по номеру */
0x20        32     agf_levels[XFS_CNUM] = 1     /* Уровень Би-дерева свободных
                                                    фрагментов с сортировкой
                                                    по размеру */
0x24        32     agf_spare1      = 0          /* запас */
0x28        32     agf_flfirst     = 0          /* Первый элемент списка
                                                    зарезервированных
                                                    свободных блоков */
0x2C        32     agf_fllast      = 3          /* Последний элемент списка
                                                    зарезервированных
                                                    свободных блоков */
0x30        32     agf_flcount     = 4          /* Количество элементов в
списке
                                                    зарезервированных
                                                    свободных блоков */
0x34        32     agf_freeblks    = 16376      /* Свободных блоков */
0x38        32     agf_longest     = 16376      /* Размер длиннейшего фрагмента
                                                    свободных блоков
                                                    в Группе Выделения*/

0x3C
}
```

Внимательный читатель уже должен заметить и загореться вопросом, и я поспешу ему ответить: Да! Действительно, ребята из SGI решили хранить список свободных блоков не в bitmap'е, а в Би-дереве с элементами "Начальный блок фрагмента свободных блоков"/"Размер фрагмента", да не в одном Би-дереве, а сразу в двух дублирующих: в одном элементы отсортированы по номеру начального блока фрагмента, а во втором - по размеру этих фрагментов свободного пространства.

Это позволяет быстро найти группу свободных блоков наиболее подходящего размера.

В примере структуры указано, что корни этих деревьев (элементы массива 0x10 *agf_roots*) находятся в блоках 1 и 2. Т.к. уровни деревьев (элементы массива 0x1C *agf_levels*) равны единице, то эти корни одновременно являются и листьями. А подробнее структура Би-дереьев будет описана ниже.

Список зарезервированных свободных блоков

Для распределителя (allocator'a) блоков XFS резервирует несколько блоков, из рассчёта максимальной высоты Би-деревьев, чтобы не дай Бог ещё и Би-деревья были фрагментированными..

Список этих зарезервированных блоков располагается со смещением **0x600** в Группе Выделения. Он представляет из себя простой массив с элементами по 32 бита каждый.

В приведённом примере в этом массиве хранится 4 элемента (0x30 *agf_flcount*) с индексами от 0 (0x28 *agf_flstart*) до 3 (0x2C *agf_fllast*).

А, вот, собственно, пример этого списка:

```
AGFL

0x00    32      = 4
0x04    32      = 5
0x08    32      = 6
0x0C    32      = 7
0x10    32      = 0
0x14    32      = 0
...
```

Информация о свободных и выделенных инф.узлах

Подобно свободным блокам, свободные и выделенные инф.узлы Группы Выделения описываются в структуре *xfs_agi* (определение см. в том же /usr/include/xfs/xfs_ag.h) со смещением **0x400**.

```
Структура xfs_agi
{
0x00        32     agi_magicnum    = "XAGI"     /* магическое число */
0x04        32     agi_versionnum  = 1          /* версия agi-заголовка */
0x08        32     agi_seqno       = 0          /* номер Группы Выделения */
0x0C        32     agi_length      = 16384      /* размер Группы Выделения */

0x10        32     agi_count       = 64         /* число выделенных инф.узлов
*/
0x14        32     agi_root        = 3          /* корень Би-дерева */
0x18        32     agi_level       = 1          /* уровень Би-дерева */
0x1C        32     agi_freecount   = 61         /* свободных инф.узлов */
0x20        32     agi_newino      = 128        /* последний созданный
                                                                  инф.узел
*/
0x24        32     agi_dirino      = -1         /* last directory inode chunk
*/

/*
 * Hash table of inodes which have been unlinked but are
 * still being referenced.
 */
0x28     64x32     agi_unlinked[64]
0x128

}
```

Инф.узлы выделяются группами по 64 штуки. Информация о занятости/свободе инф.узлов такой группы хранится всё в том же bitmap'е (64-битном числе).

Но т.к. каждая следующая группа инф.узлов вовсе не обязательно располагается за предыдущей, то эту информацию всё равно приходится хранить в Би-дереве.

Такая схема позволяет снять ограничение на количество инф.узлов присущее многим другим файловым системам. На XFS сообщение "Не достаточно места" однозначно, оно не может означать "Таблица инф.узлов переполнена" при создании файла.

Подробнее о всех Би-деревьях в следующем разделе.

Би-деревья

Итак, Би-деревья.. Если быть точнее B+-tree:
http://en.wikipedia.org/wiki/B+_tree

Ну, ладно, ладно.. Не буду вас отправлять на английскую Википедию.. Скажу пару слов по-русски.

Итак, кратко, Би-деревья - это деревья, в которых ключи объединяются в блоки по несколько штук, что значительно позволяет увеличить их эффективность на дисковых носителях.

Свойства B+-деревьев:

- Все листья располагаются на одном (самом нижнем) уровне
- Каждый блок (если только это не единственный лист-корень) должен быть заполнен (ключами) не менее чем наполовину.

Примерно так... Хотя очень не точно, но относительно XFS всё вышесказанное верно...

Общая структура Би-деревьев в XFS

Структуру заголовков блоков Би-деревьев XFS описывают структуры xfs_btree_sblock, xfs_btree_lblock (файл /usr/include/xfs/xfs_btree.h).

```
Структура xfs_btree_sblock | xfs_btree_lblock
{
0x00          32     bb_magic           /* магическое число */
0x04          16     bb_level           /* уровень блока. 0 -- для листьев. */
0x06          16     bb_numrecs         /* число записей */
0x08       32|64     bb_leftsib         /* указатель на брата
                                           по уровню слева (или -1) */
0x0C|10    32|64     bb_rightsib        /* указатель на брата
                                           по уровню справа (или -1) */
0x10|18
}
```

Да, чтобы описать две структуры одной схемой нам пришлось немного «смухлевать», но как видите они отличаются лишь размером ссылок на братьев (32 или 64 бит).

Само собой заголовок это ещё не весь блок. Основные данные следуют далее.

Для листьев всё просто: сразу же за полем *bb_rightsib* следует массив записей (фиксированного размера) в количестве *bb_numrecs* штук.

Для узлов же всё несколько сложнее. Записи здесь разделены на две части: ключ и указатель на блок более нижнего уровня (на подветку) отвечающего этой записи.

При этом ключ и указатель в общем случае имеют разный размер. И главное - ключ и указатель хранятся в раздельных массивах. Если массив ключей начинается сразу же за полем *bb_rightsib*, то откуда начинается массив указателей одному ...мда... известны..

Чтобы стать этим ...мда..., которому всё известно, нужно:

1. Взять размер блока, вычесть из него размер заголовка.
2. Взять размер ключа, прибавить к нему размер указателя.
3. Поделить результат 1 на 2 нацело.

4. Остаток деления 3 - это хвост не используемый в блоке.
5. Результат деления 3 - это максимальное число записей.
6. Умножить максимальное число записей на размер ключа.
7. Результат 6 - есть смещение начала массива указателей относительно начала массива ключей.

Далее, для каждого из Би-деревьев будем описывать

- Магическое число
- Размер указателей на братьев и подветки (32 или 64 бита)
- Структура записей листьев
- Структура ключей узлов

Тут, Вы, вероятно, спросите: а как же 64-х битность XFS, если здесь ссылки могут быть 32-х битными? Всё просто - 32-х битные ссылки используются для Би-деревьев свободных блоков и инф.узлов, которые привязаны к своей Группе Выделения, и, соответственно, это ссылки относительно Группы Выделения, а размер Группы Выделения (в блоках) - это 32-х битное число.

Би-деревья свободных блоков

- Магическое число для деревьев с сортировкой по:
  - номеру блока: "ABTB".
  - размеру фрагмента свободных блоков: "ABTC".
- Ссылки на братьев и подветви: 32 бита.

Записи листьев и ключи узлов у Би-деревьев свободных блоков имеют одну и ту же структуру *xfs_alloc_rec* (файл /usr/include/xfs/xfs_alloc_btree.h)

```
Структура xfs_alloc_rec
{
0x00          32     ar_startblock      /* Начальный блок фрагмента
                                           свободного пространства */
0x04          32     ar_blockcount      /* Размер фрагмента свободного
                                           пространства в блоках */
0x08
}
```

Однако, *ключи* узлов Би-деревьев с сортировкой по размеру фрагментов имеют перевёрнутую структуру:

```
{
0x00          32     ar_blockcount      /* Размер фрагмента свободного
                                           пространства в блоках */
0x04          32     ar_startblock      /* Начальный блок фрагмента
                                           свободного пространства */
0x08
}
```

Остаётся только удивляться почему же записи листьев в таком случае у них всё равно имеют прямую структуру...

Би-деревья свободных и выделенных инф.узлов

- Магическое число: "IABT".
- Ссылки на братьев и подветви: 32 бита.

Записи листьев описываются структурой *xfs_inobt_rec* (файл /usr/include/xfs/pwd).

```
Структура xfs_inobt_rec
{
0x00          32     ir_startino        /* Начальный инф.узел описываемый
                                                              записью */
0x04          32     ir_freecount       /* Число свободных инф.узлов */
0x08          64     it_free            /* Битовая карта инф.узлов.
                                                   1 -- значит свободен */
0x10
}
```

В структуре задан начальный инф.узел, описываемый записью. При этом количество этих инф.узлов всегда одно - 64.

Стоит отметить также, что в поле *ir_freecount*, из 32 бит, в действительности, используется только 7 (1000000b = 64d). Но создатели XFS не стали жадничать - всё для выравнивания!

Ключом в данных деревьях является единственное значение:

```
Ключ узлов Би-деревьев свободных и выделенных инф.узлов
{
0x00          32     startino           /* Начальный инф.узел описываемый
                                                            записью */
0x04
}
```

Би-деревья карты блоков

- Магическое число: "BMAP".
- Ссылки на братьев и подветви: 64 бита.

Записи листьев - структура *xfs_bmbt_rec_32* (файл /usr/include/xfs/xfs_bmap_btree.h)

```
Структура xfs_bmbt_rec_32
{
0x00          32     l0
0x04          32     l1
0x08          32     l2
0x0C          32     l3
0x10
}
```

Не правда ли забавная структура? И безликая...

Реальное объяснение всего этого безобразия (иначе, я просто не могу это назвать) будет дано в пункте «Карта блоков» следующего раздела.

Ключом в Би-деревьях карты блоков является также единственное значение:

```
Ключ узлов Би-деревьев карты блоков
{
0x00          64     startoff           /* Смещение фрагмента от начала
                                                  файла (в блоках) */
0x08
}
```

Инф.узлы

Структура инф.узлов делится на две части: общая для всех (ядро инф.узла) и зависимая от типа.

Ядро инф.узла

Ядро инф.узла - это структура *xfs_dinode_core* (файл /usr/include/xfs/xfs_dinode.h).

```
Структура xfs_dinode_core
{
0x00          16     di_magic        = "IN"     /* Магическое число */
0x02          16     di_mode         = 0x41ED   /* тип и режим доступа */
0x04           8     di_version      = 1        /* версия структуры */
0x05           8     di_format       = 1        /* формат второй части
                                                       инф.узла */
0x06          16     di_onlink       = 4        /* число ссылок (старое -
                                                       16-ти битное)
*/
0x08          32     di_uid                     /* UID (идентификатор
                                                       пользователя)
*/
0x0C          32     di_gid                     /* GID (идентификатор группы)
*/
0x10          32     di_nlink        = 4        /* Число ссылок */
0x14          16     di_projid       = 0        /* owner's project id */
0x16         8x8     di_pad[8]       = 0        /* запас на будущее */
0x1E          16     di_flushiter    = 14       /* incremented on flush */
0x20          64     di_atime                   /* последнее время доступа
                                                         (до миллисекунд)
*/
0x28          64     di_mtime                   /* последнее время изменения
                                                         (до миллисекунд)
*/
0x30          64     di_ctime                   /* время создания/модификации
                                                          инф.узла (до миллисекунд)
*/
0x38          64     di_size         = 28       /* размер файла в байтах */
0x40          64     di_nblocks      = 0        /* число блоков */
0x48          32     di_extsize                 /* basic/minimum extent
                                                       size for file
*/
0x4C          32     di_nextents                /* число фрагментов */
0x50          16     di_anextents               /* number of extents
                                                            in attribute fork */
0x52           8     di_forkoff                 /* attr fork offs, <<3 for
                                                       64b align */
0x53           8     di_aformat                 /* format of attr fork's data
*/
0x54          32     di_dmevmask                /* DMIG event mask */
0x58          16     di_dmstate                 /* DMIG state info */
0x5A          16     di_flags                   /* random flags,
                                                         XFS_DIFLAG_... */
0x5C          32     di_gen                     /* generation number */
0x60
}
```

В основном данные предоставляемые этой структурой совпадают с данными структуры *stat* выдаваемой соотсветсвующим системным вызовом, что не удивительно.

Однако, нас сейчас более всего интересует формат второй части инф.узла. Обратимся к структуре *xfs_dinode* (файл тот же):

```
Структура xfs_dinode
{
0x00        3072     di_core                    /* ядро инф.узла
                                                    (см. структуру выше) */
0x60          32     di_next_unlinked   = -1    /* agi unlinked list ptr */
0x64          xx     di_u
0x64+xx       yy     di_a
0x64+xx+yy
}
```

Так, за ядром следует некоторый указатель, который в отмонтированном состоянии похоже всегда будет равен -1.

Далее следуют два union'а. Второй из них - описывает расширенные аттрибуты. Однако, мы не будем их рассматривать и тогда окажется что размер элемента *di_a* равен нулю.

Форматы инф.узла

Мы лучше сосредоточим своё внимание на элемент *di_u* - это собственно та вторая часть инф.узла которая нас более всего интересует сейчас. Её формат зависит от значения элемента *di_format* структуры *xfs_dinode_core*, рассмотренной чуть выше. Значения *di_format* описываются перечислением (enum'ом) *xfs_dinode_fmt* (файл всё тот же - /usr/include/xfs/xfs_dinode.h):

```
Значения для di_format
{
00 =    XFS_DINODE_FMT_DEV,             /* специальный файл (устройство) */
01 =    XFS_DINODE_FMT_LOCAL,           /* директория короткого формата
                                           или символическая ссылка */
02 =    XFS_DINODE_FMT_EXTENTS,         /* карта блоков (для директорий,
                                           простых файлов и симв.ссылок) */
03 =    XFS_DINODE_FMT_BTREE,           /* Би-дерево карты блоков (для
                                              директорий и простых файлов) */
04 =    XFS_DINODE_FMT_UUID             /* MNT: di_uuid */
}
```

Специальный файл

Значение *di_format = XFS_DINODE_FMT_DEV* предназначено для специальных файлов.

Специальные файлы описываются 32-х битным числом типа *xfs_dev_t*. Оно похоже на используемый в ядре Linux *kdev_t*. Но если там MINOR число имеет 20 бит, то в XFS - 18.

Надо сказать, что этот формат явно имеет исторические причины, и даже макрос-конструктор *xfs_dev_t* из MAJOR и MINOR-чисел имеет название IRIX_MKDEV.

Локальное расположение инф.узла

Значение *di_format = XFS_DINODE_FMT_LOCAL* означает, что сам инф.узел находится прямо в хвосте его описания (со смещением 0x64).

Это применимо для символических ссылок и коротких директорий.

Тут стоит сказать об одном странном ограничении XFS: длина символических ссылок здесь не должна превышать (так же как имя файла) 255 символов. Если для имени файла такое ограничение вполне понятно, то о причинах такого же ограничения для символических ссылок остаётся только догадываться. В Ext2FS длина символической ссылки в свою очередь ограничена размером блока. (т.е. обычно это 4096 или 4095 символов)

Карта блоков

Значение *di_format = XFS_DINODE_FMT_EXTENTS* означает, что в хвосте описания инф.узла находится карта блоков.

Карта блоков - это массив значений по 128 бит каждое, описывающих один фрагмент (extent) инф.узла (директории, симв.ссылки или простого файла).

Тут настаёт время объяснения той безликой структуры данной в разделе «Би-деревья карты блоков». 80 символов ширины - конечно, узко. Но мы попробуем уложиться (не забываем, что все данные в XFS находятся в Big-Endian формате):

```
 31.30.........0 31........9.8..0 31.............0 31..21.20......0
 LLLLLLLLLLLLLLL0 LLLLLLLLLLLLLLL1 LLLLLLLLLLLLLLL2 LLLLLLLLLLLLLLL3
 ++ +------------ ----------+ +--- ---------------- -----+ +-------+
 |  |                        |                            |
 |  +-- startoff (54 бита)   +-- startblock (52 бита)     +-- blockcount (21
 бит)
 |      Смещение фрагмента       Номер блока первого          число блоков в
 |      в блоках от начала       блока фрагмента              фрагменте
 |      файла                    на диске.
 |
 +-- extent flag (1 бит)
```

Можете не сомневаться - что изображено на этом рисунке его автору совершенно ясно. Очень надеюсь, что уважаемый читатель тоже не слишком избалован качеством графических материалов, предоставляемых современным издательством, и он ещё способен ориентироваться в таких псевдографических рисунках.

Да, почему-то создатели пожадничали ещё 32 бита для этой структуры. О причинах этого нам опять же остаётся только догадываться (возможно, всё опять же для выравнивания).

Щели (Holes) в файлах

В Ext2FS, если Вы помните, в карте блоков описывается положение на диске каждого блока файла (а не по-фрагментно как в XFS). При этом sparse-блоки (т.е. блоки, которые не хранятся на диске, но считается что они заполнены нулями) там обозначаются нулём в этих картах блоков.

А XFS, благодаря избыточности информации даваемой полем *startoff*, получает возможность вообще не хранить информацию о так называемых щелях (holes) в файлах.

Т.е. если есть фрагмент с *startoff = 14* и *blockcount = 5*, вовсе не обязательно чтобы следующий фрагмент имел *startfoff = 19*. Может быть и больше (но, разумеется, не меньше).

Корень Би-дерева карты блоков

Значение *di_format = XFS_DINODE_FMT_BTREE* означает, что в хвосте описания инф.узла находится корень Би-дерева карты блоков.

Би-деревья описаны в разделе выше, но здесь для этого корня мы имеем сокращённый формат - из структуры *xfs_btree_sblock* исключается магическое число (смысла в нём в середине блока мало) и ссылки на братьев (у корня нет братьев). Остаётся (смещение указано от начала описания инф.узла):

```
 Сокращённая структура xfs_btree_sblock
 {
 0x64          16     bb_level           /* уровень блока. 0 -- для листьев. */
 0x66          16     bb_numrecs         /* число записей */
```

```
0x68
}
```

Можно отметить, что уровень этого корня (высота Би-дерева) никогда не может быть нулевым - для этого есть *di_format = XFS_DINODE_FMT_EXTENTS*.

Директории

Структура директорий в XFS - это просто песня!..

Вы, вероятно, уже обратили внимание на уровень вложенности содержания этого пункта..

Ну, так начнём наш сказ...

Короткий формат

Короткий формат - тот самый, что используется в конце описания инф.узла, когда *di_format = XFS_DINODE_FMT_LOCAL*.

Директория в этом случае имеет структуру *xfs_dir2_sf* (файл /usr/include/xfs/xfs_dir2_sf.h):

```
Структура xfs_dir2_sf
{
0x64       48|80     hdr               /* заголовок */
0x6A|0x6C     ~~     list[count]       /* элементы директории */
0x~~
}
```

Заголовок - это структура *xfs_dir2_sf_hdr* (файл тот же):

```
Структура xfs_dir2_sf_hdr
{
0x64           8     count             /* число элементов */
0x65           8     i8count           /* число элементов с
                                              8-ми байтными номерами */
0x66       32|64     parent            /* родительский инф.узел */
0x6A|0x6C
}
```

Элементы имеют структуру *xfs_dir2_sf_entry* (файл всё тот же):

```
Структура xfs_dir2_sf_entry
{
0x00           8     namelen           /* длина имени */
0x01          16     offset            /* сдвиг в одноблочной директории,
                                            кратно восьми (см. ниже) */
0x03       8xlen     name[len]         /* имя файла */
0x03+len   32|64     inumber           /* инф.узел */
}
```

В директории в коротком формате не упоминаются элементы '.' и '..'. Но вместо элемента '..' вводится поле *parent*.

Обратим Ваше внимание на ссылки *parent* и *inumber* - несмотря на неоднозначность их размера указанную в структурах, вы можете смело считать их 32-хбитными. Однако, привет счастливым обладателям Терабайтовых файловых систем - если хотя бы один элемент директории или её родитель - является инф.узлом с номером, который не укладывается в 32

бита, то все эти ссылки разом становятся 64-х битными, а драйвер файловой системы узнает об этом по ненулевому значению *i8count*.

Итак, это только присказка, сказка - впереди. Короткий формат - для коротких директорий. А что же делать с длинными? А об этом поведует следующий раздел.

Одноблочная директория

Блоки директорий в XFS - это не совсем блоки файловой системы. Они могут быть больше, о чём будет свидетельствовать не нулевое значение элемента 0xC0 *sb_dirblklog* в суперблоке.

Смещение полей следующих структур имеет смысл привязывать не только к началу блока, но и к его концу. Так договоримся, считать что 0x00 - это начало блока, 0x08 - 8 байт от начала блока, 2x00 - конец блока, 1xF8 - 8 байт от конца блока.

Структура одноблочной директории *xfs_dir2_block* описана в файле /usr/include/xfs/xfs_dir2_block.h:

```
Структура xfs_dir2_block
{
0x00         128     hdr                /* заголовок */
0x10          ~~     u[count]           /* элементы директории */
1xF8-8xN    64xN     leaf[count]        /* отсортированный по ключу хэш */
1xF8          64     tail               /* хвост, где указано число элементов
*/
2x00
}
```

Последующее описание структуры возможно имело бы смысл начать с конца (я подозреваю что драйвер файловой системы просматривает её именно так), тем не менее мы не будем переворачивать всё с ног на голову.. Пойдём попорядку.

Заголовок блока данных директории

Заголовок - структура *xfs_dir2_data_hdr* (файл /usr/include/xfs/xfs_dir2_data.h).

```
Структура xfs_dir2_data_hdr
{
0x00          32     magic        = "XD2B"      /* число элементов */
0x04        32x3     bestfree[3]                /* лучшие свободные подблоки */
0x10
}
```

Где элементы массива *bestfree* имеют структуру xfs_dir2_data_free (файл тот же):

```
Структура xfs_dir2_data_free
{
0x00          16     offset     /* смещение подблока с начала блока
                                                  (кратно восьми) */
0x02          16     length     /* длина в байтах */
0x04
}
```

Тут уважаемый читатель, вероятно, пришёл в некоторое замешательство: «что это за подблоки такие?». Всё дело в том, что длина имён файлов всех разная (IRIX - вам не DOS), также как и длина файлов. И по тем же причинам (я надеюсь, Вы прекрасно все эти причины знаете и понимаете), по которым диск стали делить на блоки, блоки директорий создатели XFS решили разделить на подблоки.

Подблок - это 8 байт. Также, как блоки в файловой системе, подблоки могут быть свободными или занятыми. Существует фрагментация свободного пространства подблоков (но разумеется не самих записей). А так как создатели XFS очень заботяться о возможности быстро находить свободный фрагмент нужного размера, то здесь помещены эти записи *bestfree*.

Надеюсь, вы теперь также прекрасно понимаете почему *offset* кратен восьми.

Элементы директории

Перейдём теперь непосредственно к этим подблокам:

В группе подблоков хранится union *xfs_dir2_data_union_t* (файл всё ещё /usr/include/xfs/xfs_dir2_data.h):

```
typedef union {
        xfs_dir2_data_entry_t   entry;
        xfs_dir2_data_unused_t  unused;
} xfs_dir2_data_union_t;
```

entry при этом описывает элемент директории, а *unused* - группу свободных подблоков. Разумеется, вы никогда не встретите в массиве *u* структуры *xfs_dir2_block*, две *unused* записи подряд (т.к. они должны быть сразу объединены).

Итак, элемент директории описывается структурой *xfs_dir2_data_entry* (файл всё тот же):

```
Структура xfs_dir2_data_entry
{
0x00          64     inumber            /* инф.узел */
0x08           8     namelen            /* длина имени */
0x09       8xlen     name[len]          /* имя */
              ~~                        /* пробел для выравнивания размера до
                                                окончания подблока */
1xFE          16     tag                /* смещение этого элемента от начала
                                                      блока (кратно 8-ми) */
2x00
```

Группа свободных подблоков описывается структурой *xfs_dir2_data_unused* (тот же файл):

```
Структура xfs_dir2_data_unused
{
0x00          16     freetag   = 0xFFFF /* инф.узел */
0x02          16     length             /* длина имени */
              ~~                        /* пробел для выравнивания размера до
                                               окончания группы подблоков */
1xFE          16     tag                /* смещение структуры от начала
                                                      блока (кратно 8-ми) */
2x00
```

В этих двух структурах 2x00 - размер структуры, который кратен восьми.

Теперь должно быть уже понятно, что за *offset* был в короткой форме директории.

В одноблочной директории уже присутствуют элементы '.' и '..'.

Хэш элементов директории

В конце блока следует хэш элементов директории. Хэш-функция - это функция libxfs_da_hashname(<строка>, <длина строки>);

Что она из себя представляет меня интересовало слабо - мне вполне хватило возможности её вызова, а потому мы сразу перейдём к описанию структуры элементов массива хэша. Структуры *xfs_dir2_leaf_entry* (файл /usr/include/xfs/xfs_dir2_leaf.h):

```
Структура xfs_dir2_leaf_entry
{
0x00          32     hashval            /* ключ */
0x04          32     address            /* смещение элемента от начала
                                                блока (разделено на 8!) */
0x08
```

Т.е. каждый такой элемент состоит из значения хэша и указателя в виде номера подблока.

Каждый такой элемент хэша сам занимает один подблок. Элементы в массиве отсортированы по возрастанию хэша.

Хвост одноблочной директории

Это структура *xfs_dir2_block_tail* (файл /usr/include/xfs/xfs_dir2_block.h):

```
Структура xfs_dir2_leaf_entry
{
0x00          32     count              /* число элементов в директории */
0x04          32     stale              /* count of stale lf entries */
0x08
```

Смысла элемента *stale* к сожалению мною найдено не было (всегда ноль?).

Если директория не помещается в один блок, то он разделяется на несколько следующим способом...

Многоблочная директория

Блоки данных директории

Итак, во-первых, из одноблочной директории исключается хэш. Полученная структура (блок данных директории) *xfs_dir2_data* (файл /usr/include/xfs/xfs_dir2_data.h) также занимает целый блок, но в отличии от блока одноблочной директории, в многоблочной директории блоков данных может быть несколько:

```
Структура xfs_dir2_data
{
0x00         128     hdr                /* заголовок */
0x10          ~~     u[count]           /* элементы директории */
2x00
}
```

Заголовок здесь - та же структура *xfs_dir2_data_hdr* (см. выше), но с магическим числом *magic = "XD2D"*.

Блоки хэша директории

Одноблочный хэш

Хэш же выделяется в отдельный блок - структуру *xfs_dir2_leaf* (файл /usr/include/xfs/xfs_dir2_leaf.h):

```
Структура xfs_dir2_leaf
{
0x00         128     hdr                 /* заголовок */
0x10        64xN     ents[count]         /* хэш всех элементов директории */
1xFC-2xN    16xM     bests[blockcount]   /* длины лучших свободных подблоков
                                             каждого из блоков данных */
1xFC          32     tail = blockcount   /* число блоков данных */
2x00
}
```

Формат элемента хэша (структура *xfs_dir2_leaf_entry*) уже была описана выше.

Элементы *bests* - это фактически максимум из трёх значений *bestfree[3]* соответсвующего блока данных.

Ну, а *tail* - это смешная структура *xfs_dir2_leaf_tail* с единственным элементом:

```
Структура xfs_dir2_leaf
{
0x00          32     bestcount           /* число блоков данных */
0x04
}
```

Действительно интересен здесь заголовок..

Заголовок блока хэша директории

Итак, заголовок структуры выше - это структура *xfs_dir2_leaf_hdr* (файл тот же):

```
Структура xfs_dir2_leaf_hdr
{
0x00          96     info                /* заголовок узла Би-дерева хэша */
0x0C          16     count               /* число элементов в блоке хэша */
0x0E          16     stale               /* count of stale entries */
0x10
}
```

Где заголовок узла Би-дерева хэша это структура *xfs_da_blkinfo* (файл /usr/include/xfs/xfs_da_btree.h):

```
Структура xfs_da_blkinfo
{
0x00          32     forw                /* ссылка на брата справа */
0x04          32     back                /* ссылка на брата слева */
0x08          16     magic = 0xd2f1      /* магическое число */
0x0A          16     pad                 /* для выравнивания */
0x0C
}
```

Указатели *forw* и *back* здесь пока равны -1.

Таким образом мы описали одноблочных хэш директории, но он может быть использован в директории лишь единожды, если же в один блок хэш не помещается то он уже превращается в полноценное Би-дерево следующим образом...

Многоблочный хэш (Би-дерево хэша)

Из одноблочного хэша исключается список лучших свободных подблоков в блоках данных. И остаётся блок хэша - структура вида:

```
Структура блока хэша
{
0x00         128     hdr                /* заголовок */
0x10        64xN     ents[count]        /* хэш части элементов директории */
2x00
}
```

Блок лучших свободных подблоков блоков данных

Да, для массива *bests[]* здесь тоже выделяется отдельный блок. Это структура *xfs_dir2_free* (файл /usr/include/xfs/xfs_dir2_node.h):

```
Структура xfs_dir2_free
{
0x00         128     hdr                /* заголовок */
0x10        16xN     bests[nused]       /* размер лучших свободных подблоков
                                                  блоков данных */
2x00
}
```

Где заголовок - это структура *xfs_dir2_free_hdr* (файл тот же):

```
Структура xfs_dir2_fre_hdr
{
0x00          32     magic = "XD2F"     /* заголовок */
0x04          32     firstdb            /* номер первого блока данных
                                           описанного в этом блоке */
0x08          32     nvalid             /* число описанных блоков данных
                                           в этом блоке */
0x0C          32     nused              /* похоже, всегда равно nvalid */
0x10
}
```

Таким образом таких блоков в одной директории тоже может быть несколько (при этом в каждом таком блоке поле *firstdb* равно сумме *nvalid* всех предыдущих блоков).

Адресация директории

Очень важная вещь, о которой, до сих пор не было сказано, но далее мы уже без этого не обойдёмся..

Итак, до сих пор мы не упомянули, где же находятся ссылки на номера блоков директории. Но, вероятно, Вы уже догадались - карты блоков директорий уже упоминались. Они точно такие же как и у простых файлов, с одним большим НО - адресация.

Если у файла совершенно ясно что такое 0-ой байт, 5-ый, 10-ый.. То у директории с этим некоторый напряг..

Хорошо, пусть блоки данных мы уложим попорядку и соответсвующим образом проадресуем, но как же блоки хэша и блоки лучших свободных подблоков?

Тут разработчики XFS, чтобы уже по адресу можно было понять является ли блок блоком данных, хэша или блоком лучших свободных подблоков решили разделить адресное пространство директорий на соответсвенно три части (по 32 Гб):

1. область блоков данных: байты с 0-го по 0x7FFFFFFF (от XFS_DIR2_DATA_OFFSET по (XFS_DIR2_SPACE_SIZE - 1))
2. область блоков хэша: байты с 0x800000000 по 0xFFFFFFFF (от XFS_DIR2_LEAF_OFFSET по (2*XFS_DIR2_SPACE_SIZE - 1))

3. область блоков лучших свободных подблоков: байты с 0x1000000000 по 0x17FFFFFFFF (от XFS_DIR2_FREE_OFFSET по (3*XFS_DIR2_SPACE_SIZE - 1))

Но это в байтах - не забывайте, что в карте блоков всё указывается в блоках (и не в блоках директорий, а в блоках файловой системы).

Узлы Би-дерева хэша

Итак, структура блока многоблочного хэша уже описана, тут нам остаётся только уточнить, что магическое число для узлов этого Би-дерева *hdr.info.magic = 0xfebe*.

А ссылки *hdr.info.forw* и *hdr.info.back* на братьев по уровню - это номера соответсвующих блоков, но в адресном пространстве директории (см. выше), т.е. номер блока файловой системы драйвер при этом получает обратившись к карте блоков.

В хэше узлов в качестве ключа указывается значение ключа самого последнего из всех его дочерних элементов.

В качестве адреса - номер дочернего блока в адресном пространстве директории.

Листья Би-дерева хэша

Магическое число для листьев Би-дерева хэша *hdr.info.magic = 0xd2ff*.

Ссылки на братьев по уровню - снова номера блоков в адресном пространстве директории.

Ключи в хэше - хэш имени соответвующего элемента директории.

Адрес - номер подблока соответвующего элемента директории, т.е. адрес элемента директории в адресном пространстве директории делённый на 8.

Всё!

О документе...

Структура Файловой Системы XFS

This document was generated using the **LaTeX**2HTML translator Version 2002-2-1 (1.70)

Copyright © 1993, 1994, 1995, 1996, Nikos Drakos, Computer Based Learning Unit, University of Leeds.
Copyright © 1997, 1998, 1999, Ross Moore, Mathematics Department, Macquarie University, Sydney.

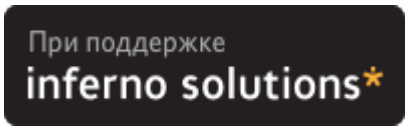
The command line arguments were:
latex2html `-html_version 4.0 -auto_link -toc_depth 7 -nonavigation -split 0 XFS.tex`

The translation was initiated by unDEFER on 2006-08-19

---

unDEFER 2006-08-19

Партнёры:



Хостинг:

Hoster.ru
хостинг провайдер

Закладки на сайте
Проследить за страницей

Created 1996-2024 by **Maxim Chirkov**
Добавить, Поддержать, Вебмастеру