author: Пешеходов А. П. aka fresco (fresco_pap@mail.ru)
mtime: 06.03.2007

Статья была опубликована в журнале "Системный администратор", № 8 (август) 2006 г.

Дизайн JFS

JFS, файловая система последнего поколения от IBM, спроектирована стройно и четко, а алгоритмы обработки данных и метаданных отличаются высокой степенью оптимизации. Однако показатели ее производительности не впечатляют. Попробуем разобраться, в чем же дело.

Истоки технологии JFS

IBM представила свою файловую систему для UNIX как JFS с первым релизом AIX v3.1 в 1990 году. Эта ФС, известная сейчас как JFS1, была основной для AIX в течение 10 лет. Пять лет спустя была начата работа по ее усовершенствованию с целью улучшения масштабируемости и реализации эффективной поддержки SMP-систем. Также ставилась задача улучшения переносимости ФС для обеспечения ее портирования и в другие операционные системы (код JFS1 был очень тесно привязан к менеджеру памяти AIX, и врядли смог бы эффективно работать в OS/2 или Linux).

Новая Journaled File System, на которой основана Linux-версия JFS, была впервые представлена с OS/2 Warp Server в апреле 1999, после нескольких лет проектирования и тестирования. В OS/2 Warp Client она была включена в октябре 2000. Параллельно с этими усилиями, в 1997 году часть разработчиков JFS вернулась в команду сопровождения AIX и занялась адаптацией кода новой JFS к этой ОС. К маю 2001 Enhanced Journaled File System (называемая JFS2) была доступна для AIX 5L. В декабре 1999 был сделан срез (snapshot) оригинального дерева кода JFS и началась работа по ее переносу в Linux.

Портирование

К декабрю 1999 уже существовали 3 перспективные ФС, только разрабатываемые или переносимые в Linux. ext2 была в процессе добавления журналирования (ext3), SGI начала перенос своей файловой системы XFS с IRIX, и Hans Reiser разрабатывал reiserfs. Ни одна из указанных ФС не была полностью функциональна в Linux в 1999, и IBM решила, что JFS является достаточно сильной технологией и добавит вес набирающему силу Linux.

Программисты из IBM посоветовались с основными разработчиками файловых систем Linux на счет возможности добавления еще одной журналируемой ФС. Одно из основных положений философии Linux: "выбор – это хорошо", поэтому идея еще одной ФС была встречена с энтузиазмом.

IBM воспользовалась поддержкой сообщества Open Source. Команда JFS последовала

совету: "release early and release often" (дословно: "выпустить рано и выпускать часто"), и привлекла множество сторонних разработчиков к переносу JFS в Linux. Всего через 3 месяца после начала работ (в феврале 2000) был выпущен первый Linux-релиз JFS. Он включал в себя функции mount/unmount и поддержку программы ls.

Рассмотрим особенности устройства и реализации JFS более подробно.

Дисковый раздел

Файловая система JFS создается на *разделе* (partition). Дисковый раздел, с точки зрения JFS, имеет следующие параметры:

- Фиксированный размер блока, PART_BSize, который может принимать значения 512, 1024, 2048 или 4096 байт. Этот параметр определяет размер наименьшего элемента обмена данными с разделом. Он соответствует размеру физического сектора того диска, на котором расположен раздел, и чаще всего равен 512 байт.
- Размер раздела, величину PART_NBlocks, содержащую количество дисковых блоков на разделе.
- Абстрактное адресное пространство: [0,(PART_NBlocks-1)] дисковых блоков.

Логический том

Для соответствия стандарту DCE DFS (Distributed Computing Environment Distributed File System – распределенная файловая система для распределенной среды вычислений) JFS отделяет понятие пула дискового пространства от понятия монтируемого поддерева именованных объектов файловой системы. Дисковый раздел представлен в JFS *логическим томом* (aggregate, в действительности, означает "совокупность", "целостность", но для удобства мы будем называть его томом), а совокупность файлов и каталогов именуется *файлсетом* (fileset – набор фалов). Существует в точности один логический том на физическом разделе, файлсетов же может быть несколько (в Linux эта возможность не поддерживается из-за ограничений интерфейса VFS). В первом релизе JFS поддерживала только один набор файлов на том, однако все метаданные были спроектированы для более общего случая.

Логический том состоит из следующих частей:

- 32 Kb неиспользуемого дискового пространства в начале.
- Первичный и вторичный суперблоки, содержащие глобальную информацию о данном томе. Вторичный суперблок есть точная копия первичного, положение обоих на диске фиксировано. Структура суперблока описана в jfs_superblock.h:

```
/*
 *    Суперблок логического тома
 */
```

```
struct jfs_superblock {
      char s_magic[4];        /* Magic-номер */

      __le32 s_version;       /* Версия */

      __le64 s_size;          /* Размер тома в физических блоках,
                               * для VFS - количество блоков */

      __le32 s_bsize;         /* Размер блока тома в байтах,
                               * для VFS - размер фрагмента */

      __le16 s_l2bsize;       /* Двоичный логарифм s_bsize */

      __le16 s_l2bfactor;     /* Двоичный логарифм (s_bsize/s_pbsize) */

      __le32 s_pbsize;        /* Размер физического блока в байтах */

      __le16 s_l2pbsize;      /* Двоичный логарифм s_pbsize */

      __le16 pad;             /* Дополнение (для выравнивания) */

      __le32 s_agsize;        /* Размер allocation-группы в логических
                               * блоках */

      __le32 s_flag;          /* Атрибуты тома (см. jfs_filsys.h) */

      __le32 s_state;         /* Состояние JFS (mount/unmount/recovery),
                               * см. jfs_filsys.h  */

      __le32 s_compress;      /* >0 если задействовано сжатие данных */

      pxd_t s_ait2;           /* Первый экстент вторичной таблицы inode
                               * тома */

      pxd_t s_aim2;           /* Первый экстент вторичной карты inode
                               * тома */

      __le32 s_logdev;        /* Адрес устройства, на котором расположен
                               * журнал */

      __le32 s_logserial;     /* Номер mount-сессии */

      pxd_t s_logpxd;         /* Встроенный экстент журнала */

      pxd_t s_fsckpxd;        /* Встроенный экстент рабочего пространства
                               * для fsck */

      struct timestruc_t s_time;    /* Время последнего обновления */

      __le32 s_fsckloglen;    /* Количество блоков, зарезервированных для
                               * служебного журнала fsck. */

      s8 s_fscklog;           /* Определяет, какой из сервисных логов fsck
                               * наиболее свежий:
                               *    0 - в логах еще нет данных
                               *    1 - первый
                               *    2 - второй */

      char s_fpack[11];       /* Имя тома. Должно быть 11 байт длиной для
                               * соответствия требованиям к загрузочному
                               * сектору OS/2. Используется только если
                               * s_version == 1 */

      /* Параметры */
      __le64 s_xsize;
```

```
        pxd_t s_xfsckpxd;
        pxd_t s_xlogpxd;

        char s_uuid[16];        /* 128-битный uuid тома */
        char s_label[16];       /* Метка тома */
        char s_loguuid[16];     /* 128-битный uuid устройства с журналом */
        };
```

- Два экземпляра таблицы inodes тома, содержащей inodes, описывающие глобальные управляющие структуры тома. Логически это просто массивы inodes. Т.к. том не имеет структуры каталога, объекты, описываемые этими inodes, не видны в пространстве имен какого-либо файлсета.
- Две копии карты размещения inodes, описывающей таблицу inodes тома.
- Карта размещения блоков, описывает структуры, управляющие выделением и освобождением блоков тома.
- Рабочее пространство для fsck, необходимое для отслеживания выделения блоков тома. Оно необходимо потому, что JFS поддерживает очень большие тома, и данные о выделенных/свободных блоках могут не поместится в памяти. Это пространство описывается суперблоком. Один байт необходим для описания 8 блоков (по 1 биту на блок). Рабочее пространство fsck всегда расположено в конце тома.
- Встроенный журнал, обеспечивающий место для журналирования изменений метаданных тома. Описывается суперблоком и всегда расположено за рабочим пространством fsck.

Состав таблицы inodes тома таков:

- Inode 0 зарезервирован.
- Inode 1 описывает все дисковые блоки тома, включая карту inodes. Это рекурсивное представление, при котором сам inode расположен в файле, который он описывает. Очевидная трудность рекурсивного представления преодолевается здесь путем размещения по крайней мере первого inode тома в фиксированном месте (через 4 Kb после первичного суперблока тома). Таким образом, JFS может просто отыскать inode 1, а от него и остальные inodes таблицы – следуя по B+ дереву в inode 1. Для репликации таблицы inodes тома JFS также должна находить копию inode 1 – для достижения оставшейся вторичной таблицы. Для этих целей в суперблоке есть поле, содержащее дескриптор экстента, описывающий положение inode 1 вторичной таблицы.
- Inode 2 описывает карту размещения блоков тома.
- Inode 3 описывает встроенный журнал (когда ФС смонтирована). Дисковая копия этого inode не указывает на какие-либо данные, его поля заполняются только в in-memory образе.
- Inode 4 описывает файл плохих блоков, найденных при форматировании тома.
- Inodes с 5-го по 15-й зарезервированы.
- Начиная с 16-го, существует по 1 inode на каждый файлсет. Эти inodes описывают управляющие структуры, представляющие набор файлов. С добавлением новых

наборов файлов к тому таблица inodes тома может расти.

Allocation groups

Allocation groups (AGs, группы размещения) разбивают пространство тома на части и позволяют политике выделения ресурсов JFS использовать хорошо известные методы для достижения высокой производительности ввода/вывода. Во-первых, JFS пытается группировать вместе дисковые блоки и inodes для связанных данных. Во-вторых, ФС старается распределить несвязанные данные по всему тому для согласования размещения. AGs в пределах тома идентифицируются отсчитываемым от нуля индексом, именуемым номером AG.

Размеры AGs выбираются таким образом, что бы они могли вмещать достаточно большие объекты. Для минимизации количества изменений, которые необходимо внести при расширении или усечении тома (имеется в виду операция resizefs), максимальное количество AGs ограничено 128. С другой стороны существует ограничение на минимальный размер одной AG – в 8192 блоков тома. Число, выражающее размер AG в dmap-страницах (4 Кб, от disk map – дисковая карта), должно быть степенью двойки. Выбранный исходя из всех этих условий размер записывается в специальное поле суперблока тома.

Том с размером, не кратным размеру AG, будет содержать одну частичную группу размещения. Она обрабатывается так же, как и обычная AG, а несуществующие физически блоки в карте выделения будут помечены как занятые.

Файлсеты

Как уже говорилось, файлсет есть просто набор именованных файлов и каталогов. Он полностью содержится в пределах одного тома. Заметьте также, что на одном томе может существовать несколько файлсетов; в этом случае все они разделяют общий пул свободных дисковых блоков тома.

Каждый файлсет располагает собственной таблицей inodes и картой их размещения, которая содержит данные о положении inodes на диске. "Супер-inode", описывающий эту карту и другую глобальную информацию о файлсете, расположен в таблице inodes тома и был описан выше.

Inodes в таблице файлсета имеют следующие значения:

- Нулевой inode зарезервирован.
- Inode 1 содержит дополнительную информацию о файлсете, которая не поместилась в структурах, описываемых inodes тома.
- Inode 2 описывает корневой каталог данного файлсета. Заметьте, что JFS соблюдает общее для файловых систем UNIX соглашение, когда inode за номером 2 описывает корневой каталог.

- Inode 3 описывает ACL-файл.
- Начиная с четвертого, следуют inodes для обычных объектов файловой системы – файлов и каталогов.

Структуры данных

Каждый объект JFS описывается inode. Inode содержит специфичную для объекта информацию, например временные штампы и тип файла (регулярный, каталог, или др.). Он также "содержит" B+ дерево для отслеживания положения экстентов. Примечательно, что все метаданные JFS (за исключением суперблока) представлены в виде файлов. Благодаря использованию структуры inode для описания размещения этих данных, их положение на диске может быть изменено в новых версиях JFS без внесения изменений в код.

Экстенты

Файл размещается на диске в виде группы экстентов. Экстент может иметь длину от 1 до (2^24-1) блоков тома; следовательно, большие экстенты охватывают несколько AGs. Экстенты индексируются в B+ дереве для улучшения производительности операций поиска и вставки.

Для однозначного определения экстента необходимо знать всего 2 параметра: номер его стартового блока и длину.

Основанная на экстентах ФС, предоставляющая пользователю возможность задавать размер блока, по идее, почти не подвержена внутренней фрагментации. Пользователь может сконфигурировать том с маленьким размером блока (минимально – 512 байт) для уменьшения внутренней фрагментации при работе с большим количеством мелких файлов.

Вообще говоря, драйвер JFS старается размещать файлы в как можно меньшем количестве экстентов, что позволяет формировать длинные I/O запросы к диску, которые (как на запись, так и на чтение) выполняются гораздо более эффективно, чем группа коротких. Однако это не всегда возможно. К примеру, copy-on-write клонирование сегмента файла вызовет деление протяженного экстента на несколько более коротких. Другой пример – ограничение размера экстента. Он может быть лимитирован при обработке сжатых файлов – т.к. драйвер должен прочитать в память весь экстент и разжать его. Имея в своем распоряжении ограниченный объем памяти, драйвер JFS должен быть уверен, что разжимаемые данные не выйдут за пределы доступного региона.

Утилита для дефрагментации избавляет пользователя от роста внешней фрагментации, которая возникает из-за динамического выделения/освобождения экстентов переменной длины. Результатом этого процесса является появление разбросанных по всему диску свободных экстентов разного размера. Утилита дефрагментации производит объединение множества мелких свободных экстентов в один большой.

Inodes

Дисковый inode JFS занимает 512 байт и содержит 4 базовых группы данных (четверти). Первая группа хранит POSIX-атрибуты JFS-объекта, вторая – дополнительные атрибуты (данные для работы с VFS, специфичную для ОС информацию и заголовок B+ дерева). Третья группа содержит либо дескриптор экстента корневого узла B+ дерева, либо встроенные данные, четвертая – расширенные атрибуты, встроенные данные или дополнительные дескрипторы экстентов. Структура дискового inode определена в файле jfs_dinode.h как структура dinode.

```
/*
 *    on-disk inode : 512 bytes
 *
 *    note: align 64-bit fields on 8-byte boundary.
 */
struct dinode {

      /*
       *    1. Базовая четверть - POSIX атрибуты. 128 байт.
       */

      __le32 di_inostamp;     /* признак принадлежности inode к конкретному
                               * файлсету */

      __le32 di_fileset;      /* номер файлсета */
      __le32 di_number;       /* номер inode */
      __le32 di_gen;          /* номер поколения inode */

      pxd_t di_ixpxd;         /* дескриптор inode-экстента */

      __le64 di_size;         /* размер объекта */
      __le64 di_nblocks;      /* количество выделенных блоков */

      __le32 di_nlink;        /* счетчик ссылок на объект */

      __le32 di_uid;          /* UID владельца */
      __le32 di_gid;          /* GID владельца */

      __le32 di_mode;         /* атрибуты, формат, разрешения */

      struct timestruc_t di_atime;  /* время последнего доступа */
      struct timestruc_t di_ctime;  /* время последней модификации атрбутов */
      struct timestruc_t di_mtime;  /* время последней модификации данных */
      struct timestruc_t di_otime;  /* время создания */

      dxd_t di_acl;           /* ACL-дескриптор */

      dxd_t di_ea;            /* EA-дескриптор */

      __le32 di_next_index;   /* следующий доступный индекс в dir_table */

      __le32 di_acltype;      /* тип ACL */

      /*
       *    2,3,4. Четверти расширений.
       */
      union {
```

```
        struct {
              struct dir_table_slot _table[12];   /* Таблица directory-
                                                   * слотов. */

              dtroot_t _dtroot;              /* Корень дерева каталогов */
        } _dir;

        struct {
              union {
                    u8 _data[96];            /* не используется */
                    struct {
                          void *_imap;       /* не используется */
                          __le32 _gengen;    /* generator */
                    } _imap;
              } _u1;
              union {
                    xtpage_t _xtroot;        /* корень дерева экстентов */
                    struct {
                          u8 unused[16];     /* не используется */
                          dxd_t _dxd;        /* EA-дескриптор */
                          union {
                                __le32 _rdev;           /* [minor:major] */
                                u8 _fastsymlink[128];   /* имя для symlink */
                                } _u;
                          u8 _inlineea[128];      /* встроенные EAs */
                          } _special;
                    } _u2;
              } _file;
        } u;
  };
```

В JFS inodes размещаются динамически, что дает несколько преимуществ по сравнению со статическим подходом. Дисковый блок под inode может быть размещен по любому адресу, что развязывает номер inode с его координатами. Это упрощает изменение размера тома. Inodes могут перемещены без их перенумерации, что избавляет JFS от необходимости вникать в структуру каталогов при перемещении inode. Эта развязка также необходима для поддержки клонирования файлсетов, обязательной для соответствия стандарту DFS.

С другой стороны, это порождает и проблемы. При статическом размещении геометрия файловой системы явно описывается положением inodes на диске, с динамическим же размещением требуются отдельные картирующие структуры. Из-за накладности репликации этих структур, дизайнеры JFS решили смирится с риском потери метаданных. Однако JFS журналирует B+ деревья, что позволяет в случае сбоя без особого труда отыскивать эти карты.

На диске inodes размещаются в виде inode-экстентов, представляющих собой массивы inodes. По определению такой экстент содержит 32 inodes и, следовательно, занимают 16 Kb. При размещении нового inode-экстента он не заполняется нулями. Для того, что бы проверить, используется ли данный inode, fsck смотрит на его счетчик ссылок.

Для поиска inodes на диске служат карты размещения inodes (см. ниже).

B+ деревья экстентов

Дескриптор выделенного экстента (структура xad_t из файла jfs_xtree.h, далее просто XAD) описывает экстент и имеет 2 дополнительных поля для использования в файлах: offset – логическое смещение от начала файла, и поле флагов:

```
/*
 *    Дескриптор выделенного экстента (XAD)
 */
typedef struct xad {
      unsigned flag:8;        /* флаги */
      unsigned rsvrd:16;      /* зарезервировано */
      unsigned off1:8;        /* смещение в блоках */
      __le32 off2;            /* смещение в блоках */
      unsigned len:24;        /* длина в блоках */
      unsigned addr1:8;       /* адрес в блоках */
      __le32 addr2;           /* адрес в блоках */
} xad_t;
```

- *flag* – 8-битное поле, содержащее различные флаги, например бит copy-on-write, если экстент размещенный, но не записываемый, данные для компрессии, и т.д..
- *rsrvd* – 16-битное поле, зарезервированное для будущего использования. Всегда ноль.
- *off1,off2* – 40-битное поле, выражающее логическое смещение первого блока экстента от начала файла в блоках
- *len* – 24-битное поле, содержащее длину экстента в блоках.
- *addr1,addr2* – 40-битный адрес экстента

XAD описывает 2 абстрактных диапазона: физический, относительно всего тома, и логический – относительно файла, которому принадлежит экстент. JFS поддерживает одну generic-структуру B+ дерева для всех индексируемых объектов. Меняется только формат листовых узлов. В дереве дескрипторов выделенных экстентов (свое для каждого файла) ключами являются их логические смещения.

Нижняя часть второй четверти дискового inode содержит маркер, который указывает, что хранится во второй половине inode, которая может содержать встроенные сырые данные – если файл достаточно мал, или корневой узел B+ дерева экстентов – если файл в inode не помещается. Заголовок узла описывает, сколько XAD'ов использовано и сколько еще доступно. Вообще говоря, если данные файла расположены в восьми или менее экстентах, то корень дерева будет содержать просто до 8 XAD-структур, представляя из себя листовой узел. Иначе эти восемь XAD'ов будут указывать на дополнительные листовые или внутренние узлы дерева.

Как только 8 XAD-структур в третьей четверти inode заполнятся, драйвер JFS сделает попытку разместить дополнительные XAD'ы в четвертой четверти. Если бит INLINEEA (см. ниже) в поле di_mode дискового inode установлен, последняя его четверть доступна.

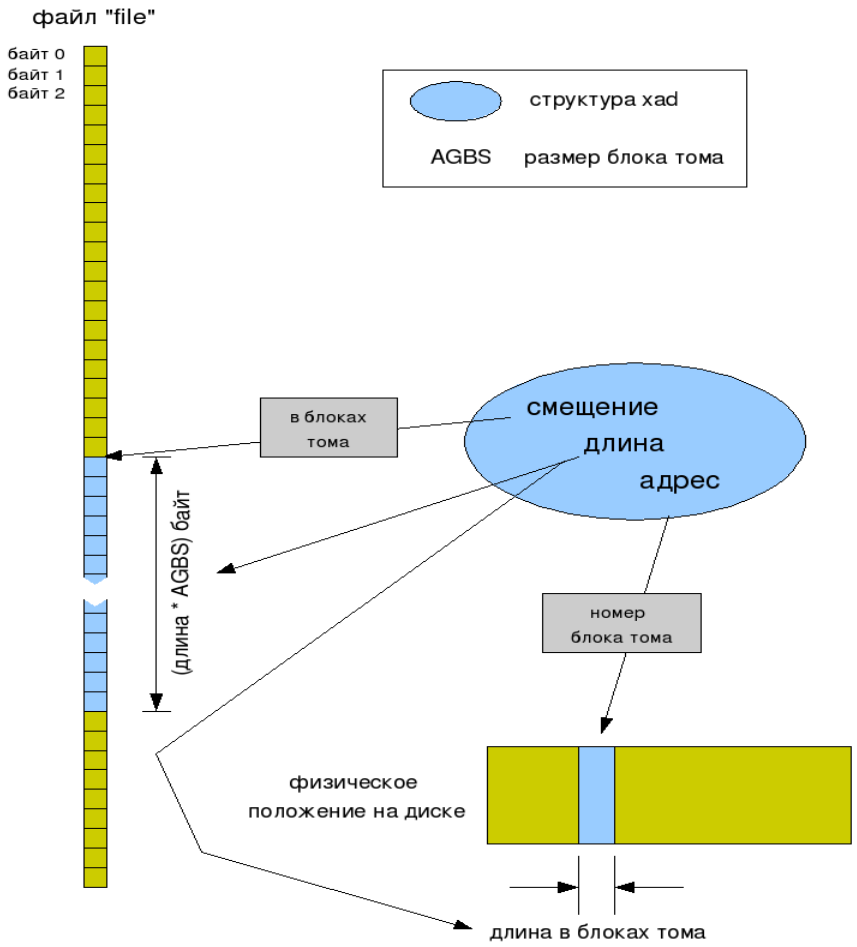

Рис. 1. XAD описывает 2 абстрактных диапазона.

Когда все возможные XAD-структуры в inode использованы, B+ дерево разбивается. JFS выделяет 4 Кб дискового пространства для листового узла дерева (в терминологии JFS любой регион дискового пространства размером в 4 Кб именуется *страницей;* в дальнейшем вы увидите, что, не смотря на обилие "промежуточных" сущностей – типа блока раздела и блока тома, JFS оперирует в основном со страницами; вероятно, это связано с особенностями подсистемы управления памятью AIX и OS/2), который является просто массивом XAD-структур с заголовком, содержащим указатель на первый свободный XAD. Далее 8 XAD'ов копируются из inode в лист, заголовок инициализируется указателем на 9-й XAD. Затем JFS обновляет корень дерева – первую XAD-структуру в inode. Она теперь будет указывать на вновь размещенный лист. Также обновляется заголовок корневого узла – ведь теперь он содержит только один XAD. Маркер во второй четверти inode теперь показывает, что inode содержит чистый корень B+дерева.

По мере добавления новых экстентов к файлу, их дескрипторы будут вставляться в тот же самый листовой узел – пока он не заполнится. Когда это произоисходит, выделяются еще 4 Кб дискового пространства для нового листа, а в inode добавляется еще один XAD, указывающий на этот узел. Так продолжается до тех пор, пока все 8 XAD-структур в корневом узле (в inode) не будут использованы. В этом случае размещается внутренний узел дерева размером 4 Кб, в который буду скопированы 8 XAD'ов из inode. Корневой узел же полностью создается заново и на выходе содержит лишь один XAD, указывающий на вновь

размещенный внутренний узел.

В файле jfs_xtree объявлена структура xtpage, описывающая заголовок корня B+дерева. В jfs_btree.h есть структура btpage, формат которой соответствует виду заголовков внутренних и листовых узлов дерева.

Карта выделения блоков

Карта выделения блоков используется для отслеживания состояния (выделен/свободен) блоков тома. Т.к. все файлсеты используют общий пул дисковых блоков, они используют и общую карту.

Логически эта карта представляет собой файл, описываемый вторым inode тома. При форматировании раздела выделяется место сразу под всю карту, она может менять свой размер только в случае увеличения/усечения тома.

Карта состоит из трех типов 4-килобайтных страниц – контрольная bmap-страница (от block map – блочная карта), контрольные dmap-страницы и сырые dmap-страницы. Сырые dmap-страницы содержат по одному биту на каждый блок; если бит установлен – блок выделен, если сброшен – свободен. Структуры всех типов страниц определены в jfs_dmap.h. Не трудно догадаться, что контрольная bmap-страница представляет собой корень дерева, контрольные dmap'ы – его внутренние узлы, а сырые dmap'ы – листья. Высота дерева зависит от размера тома. По умолчанию оно имеет 3 уровня (и может описывать 2^43 блоков), однако, если таковое их количество не требуется, inode блочной карты будет описывать разреженный файл с "дырами" на месте первой страницы каждого из неиспользуемых уровней (напомню, что "дырой" называется экстент, не имеющий дискового воплощения, т.е. поле адреса такого экстента будет неопределено).

JFS использует особую стратегию фиксации для обеспечения надежного внесения обновлений в управляющие структуры. Для этого JFS поддерживает 2 битовые карты в каждой dmap-странице – рабочую и постоянную. Рабочая карта отражает текущее состояние блоков, а постоянная – зафиксированное. При освобождении блока первой обновляется постоянная карта, при выделении – рабочая.

Выделение inodes

При динамическом размещении inodes их номера более не привязаны к положению на диске, и это порождает ряд проблем, для решения которых было необходимо ввести дополнительные структуры данных:

- Прямой поиск: по номеру inode найти его на диске (например, при открытии файла).
- Обратный поиск: по номеру блока найти ближайшие свободные inodes (при размещении новых inodes).
- Поиск свободных номеров inodes: для размещения нового inode-экстента

необходимо найти 32 последовательных не занятых номера inode.

Карта размещения inodes

Карта размещения inodes решает проблему прямого поиска. Каждый файлсет тома поддерживает собственную карту, которая является динамическим массивом групп размещения inodes (Inode Allocation Group – IAG). Эта карта физически расположена в файле, описываемым "супер-inode" из таблицы тома.

Каждая IAG имеет 4 Кб в размере и описывает 128 inode-экстентов. Т.к. каждый inode-экстент содержит 32 inodes, IAG фактически адресует 4096 inodes. IAG может существовать в любом месте тома, однако все описываемые ею экстенты должны принадлежать одной AG, и эта группа прикрепляется к AG до тех пор, пока хотя бы один ее inode используется. Как только все inodes освобождены, новые inodes могут размещаться в любой другой AG. Формат IAG определен как struct iag в jfs_imap.h.

Первая 4-килобайтная страница карты является контрольной и содержит общую информацию о ней. Логически карта размещения inodes является массивом структур типа struct iag.

Физически карта inodes расположена в файле, описываемом специальным inode из таблицы файлсета. Страницы этого файла размещаются и освобождаются по необходимости, с применением стандартного механизма индексирования через B+ дерево.

Механизм журналирования карт размещения inodes аналогичен механизму защиты битовых карт блоков и предусматривает наличие рабочего и постоянного экземпляров карты.

Каждая страница IAG имеет контрольную секцию, содержащую обобщающую карту inodes. Она используется для улучшения производительности процесса поиска Каждый бит такой карты отображает состояние 32 inodes и сбрасывается, если все они свободны.

Кроме того, IAG содержит 128 дескрипторов inode-экстентов, на которые также имеется обобщающую карта. Один ее бит отображает состояние одного inodes-экстента и будет сброшен, если экстент полностью свободен.

Список свободных inodes AG

Этот список решает проблему обратного поиска. Т.к. количество AGs на томе ограничено и заранее известно, то и количество заголовков этих списков также предопределено. Заголовок списка свободных inodes содержится в контрольной (первой) странице карты размещения inodes. I-тый элемент этого массива будет заголовком для двусвязного списка всех IAGs, имеющих свободные inodes. Номер IAG используется в качестве индекса, значение -1 указывает на конец списка. Каждая контрольная страница содержит указатели на голову и хвост списка.

Вставка и удаление из списка свободных inodes AG происходит в его голове. Вставка происходит, например, при размещении нового inode-экстента или при удалении inode из полностью занятого экстента. В случае, когда все inode-экстенты конкретной IAG заполняются, она удаляется из списка.

Этот список не журналируется. В случае сбоя он реконструируется fsck. Структура списка определена как struct dinomap в jfs_imap.h.

Список свободных экстентов AG

Этот список помогает решить проблему обратного поиска и поиска свободных номеров inodes. Каждый файлсет имеет свой список свободных inode-экстентов на каждую AG. Количество заголовков этих списков, хранящихся в контрольной странице карты размещения inodes, ограничено количеством AGs. I-тый элемент списка будет заголовком для двусвязного списка всех IAGs, имеющих свободные inode-экстенты. Номер IAG используется как индекс в этом массиве, -1 указывает на конец списка.

Когда все inodes в экстенте удаляются, его дисковые блоки освобождаются, а номер IAG, содержащей этот экстент, вставляется в голову списка свободных inode-экстентов. При размещении новой IAG происходит тоже самое. Когда все inode-экстенты в IAG занимаются, эта IAG удаляется из списка. Если все inode-экстенты в IAG освобождаются, IAG вырезается из этого списка и вставляется в список свободных IAGs (см. ниже).

Эта таблица не журналируется, ее формат определен в структуре dinomap в файле jfs_imap.h.

Список свободных IAGs

Эта таблица решает проблему поиска свободных номеров inodes. Ее структура боле проста: она содержит просто список номеров полностью свободных IAGs. Его заголовок является полем в inode тома, который описывает данный файлсет.

IAG Free next

Этот счетчик помогает решить проблему поиска свободных номеров inodes и позволяет JFS находить номер для вновь размещаемой IAG. Этот счетчик расположен в контрольной странице карты размещения inodes. Однажды размещенная IAG никогда не удаляется.

"Супер-inodes"

Inode тома, описывающие конкретные файлсеты являются inodes особого типа и содержат специфическую информацию в последних двух четвертях вместо нормальных для inode данных. Они отслеживают дисковое местоположение карты размещения inodes в своем

B+ дереве.

Файл

Файл представлен inode, содержащим корень B+ дерева, в котором, по логическому смещению от начала файла, проиндексированы экстенты, содержащие данные этого файла.

Символическая ссылка

Символическая ссылка представлена inode, поле di_mode которого содержит установленный флаг S_IFLNK. Полный путь к файлу, на который указывает ссылка, хранится прямо в inode. Если строка с путем в inode не помещается, под нее выделяется один или несколько экстентов, обработка которых происходит по стандартному для регулярных файлов сценарию.

Каталог

Каталог – это журналируемый файл с метаданными JFS. Он состоит из элементов каталога, которые описывают содержащиеся в нем объекты. Каждый такой элемент связывает имя объекта с соответствующим ему номером inode. Для увеличения производительности поиска по каталогу элементы проиндексированы по имени с помощью B+ дерева.

Поле di_size в описывающем каталог inode содержит количество листовых узлов B+дерева, которым проиндексированы элементы каталога. Если все содержимое каталога умещается в одном листе, то он будет полностью хранится в inode, а di_size=256.

Каталог в JFS не хранит на диске 2 специфичных элемента – указатели на себя (".") и на родительский каталог (".."). Они представлены полями в самом inode: указатель на себя – это просто номер inode каталога, а указатель на родителя – поле idotdot, содержащее структуру dtroot, определенную в файле jfs_dtree.h.

Inode каталога содержит корень своего B+ дерева, подобно регулярному файлу. Ключем в этом дереве является хэш имени объекта, листовые узлы содержат элементы каталога, состоящие из полного имени и номера inode.

Т.к. элементы каталога могут быть разного размера, JFS нуждается в особой схеме их обработки. Для их хранения используются т.н. directory-слоты – структуры фиксированного размера, имеющие поля для номера inode и имени. Если имя в один слот не помещается, выделяется следующий – с тем же номером inode. Внутренние узлы дерева состоят из двух частей: массива сжатых суффиксов имен (используется как ключ) и таблицы простых дескрипторов экстентов (указывают на узлы дочерних уровней). Листовые узлы хранят массивы полных имен (ключ) и номеров inodes(данные).

Внутренние и листовые узлы B+ дерева каталога занимают страницу в 4 Кб. Т.к. многие каталоги не столь велики, подобная схема хранения приводит к потере дискового пространства. Для решения этой проблемы была введена следующая схема выделения:

1. Изначально элементы каталога хранятся в области встроенных данных в inode.
2. Когда эта область переполняется, JFS выделяет под листовой узел 1 блок тома.
3. Когла этот блок переполняется, JFS пытается увеличить его размер в 2 раза (если он еще не равен 4 Кб); если места вокруг текущего экстента не хватает, выделяет новый экстент и копирует в него данные, не забывая внести изменения в родительский узел.
4. Если этот новый экстент вновь заполняется, и его размер еще не достиг 4 Кб, повторяется шаг 3. Если нет – размещается новый лист. Все выделяемые в последствии листовые узлы будут размером 4 Кб.
5. Когад все элементы в листовой странице освобождаются, эта страница удаляется из дерева. Содержимое каталога будет упаковано обратно в inode только когда все его элементы будут удалены.

ACL

ACL's, связанные с каждым inode в JFS, предстваляют различные свойства объекта, например разрешения, идентификаторы пользователя или группы. ACL-поля в inode из таблицы тома тома игнорируются.

Несмотря на то, что фиксированных требований к дисковому или in-memory формату списков контроля доступа не существует, JFS использует структуру, определенную стандартом DFS. ACL органичен в размере и должен помещаться в 8-килобайтную структуру dfs_acl.

Любой объект JFS может быть ассоциирован с ACL, который будет управлять дискретным доступом к этому объекту. Каталоги могут иметь до двух ACLs, которые инициализируются во время создания объекта: ACL каталога и файловый ACL. Если есть, файловый ACL будет применяться при доступе к любому файлу этого каталога.

Для хранения массива ACL's JFS использует специальный файл (в каждом файлсете), котрый структурирован так: за каждыми 8 Мб ACL's следует 4-килобайтная битовая карта занятости ACL's. ACL-файл журналируется.

Расширенные атрибуты

Расширенные атрибуты (Extended Attributes – EAs) – это generic-механизм хранения и доступа к данным, прикрепленным к объектам JFS. EAs хранятся непрерывно в пространстве расширенных атрибутов (Extended Attribute Space – EAS) как определено EA-дескриптором в inode объекта JFS. EA-дескриптор – это просто дескриптор экстента, описанный в файле jfs_types.h, в структуре dxd_t.

EA может храниться в области встроенных данных в inode, или в отдельном экстенте. Поле flags в дескрипторе EA имеет индикатор способа хранения. Т.к. область встроенных данных inode также может быть использована под дополнительные xad'ы, поле di_mode в inode будет индикатором доступности этого пространства. INLINEEA бит будет установлен, если оно доступно.

Элемент EA содержит имя атрибута и его значение. Для доступа к конкретному атрибуту JFS просто линейно ищет его в экстенте. Расширенные атрибуты не журналируются (если они хранятся не в inode), но пишутся на диск синхронно.

Потоки

Потоки (streams) в JFS используются для прикрепления дополнительных именованных данных к файлам и каталогам. Из-за ограничений интерфейса VFS, в Linux-версии JFS потоки не поддерживаются и здесь упоминаются только для полноты картины.

Вторая четверть дискового inode имеет поле для хранения дескриптора потока. Т.к. количество прикрепленных к объекту потоков может меняться, дескриптор потока содержит просто номер inode, описывающий файл со списком номеров inodes второго уровня. Эти inodes описывают непосредственно данные потока.

Потоки не журналируются.

Заключение

Архитектура и реализация JFS отличаются большим своеобразием, на фоне других файловых систем Linux она выглядит "белой вороной". Это обусловлено тем, что JFS изначально проектировалась для очень далекой от UNIX OS/2. Вероятно, IBM не смогла до конца решить одну из поставленных при разработке второй версии этой ФС задач, – создание хорошо переносимой ФС, – в среде Linux ее производительность не идеальна.

Однако, несмотря на некоторые недочеты в работе, JFS была и остается очень стабильной файловой системой, а ее дизайн представляет чисто академический интерес – как и устройство любого программного продукта, сработанного в недрах "голубого гиганта".

Источники

1. www.ibm.com/developerworks/
2. www.jfs.sourceforge.net/
3. Исходные тексты драйвера JFS для Linux-2.6.16

Свежую версию этого документа, а также аналогичные по тематике статьи и переводы можно найти на www.filesystems.nm.ru