восстановление удаленных файлов под BSD

крис касперски ака мыщъх ака souriz aka nezumi aka elraton aka laoshu, no email

какой обладатель BSD не удалял файлов за которые потом был готов повеситься? особенно это легко сделать из командной строки, когда лишний пробел или символ звездочки трут все подчистую. а ведь xBSD это одна из наиболее популярных систем, под которой вращается множество серверов и рабочих станицей. разрушения данных случаются постоянно. хотите узнать как этому противостоять?

введение

xBSD поддерживает множество файловых систем: FAT16/32, ext2fs/ext3fs, ISO 9660, UDF, NFS, SMBFS, NTFS, ReiserFS, XF, AFS, LFS, но основной системой, устанавливаемой по умолчанию была и остается UFS/UFS2.

Многие коммерческие UNIX'ы также используют либо саму UFS, либо нечто очень на нее похожее. В противоположность ext2fs, исхоженной вдоль и поперек, UFS (равно как и ее наследница — FFS) практически недокументированна и крайне поверхностно описана в доступной литературе. Единственным источником информации становятся исходные тексты, в которых не так-то просто разобраться!

Существует множество утилит, восстанавливающих уничтоженные данные (или во всяком случае пытающихся это делать), но на проверку все они оказываются неработоспособными (или обнаруживает не все файлы), что в общем-то и неудивительно, поскольку автоматическое восстановление удаленных файлов под UFS невозможно в принципе. Тем не менее, это достаточно легко сделать вручную, если, конечно, знать как.

Не пугайтесь! Это намного надежнее, интереснее и проще, чем может показаться на неискушенный взгляд! Сейчас мы погрузимся в мир файловой системы UFS и посмотрим какие шестеренки приводят ее в движение.

немного истории

UFS расшифровывается как UNIX File System и ведет свою историю от S5 FS — самой первой файловой системы, написанной для UNIX в далеком 1974 году. S5 FS была крайне простой и неповоротливой (по некоторым данным 2%-5% от "сырой" производительности голого диска), но понятия суперблока (super-block), файловых записей (inodes) и блоков данных (blocks) в ней уже существовали.

В процессе работы над дистрибутивом 4.2 BSD, вышедшим в 1983 году, ординальная файловая система претерпела некоторые улучшения. Были добавлены длинные имена, символические ссылки и т. д. Так родилась UFS.

В 4.3 BSD, увидевшей свет уже в следующем году, улучшения носили намного более радикальный, если не сказать революционный, характер. Появились концепции фрагментов (fragments) и групп цилиндров (cylinder groups). Быстродействие файловой системны существенно возросло, что и определило ее название FFS – Fast File System (быстрая файловая система).

Все последующие версии линейки 4.x BSD прошли под знаменем FFS, но в 5.x BSD файловая система вновь изменилась. Для поддержки дисков большого объема ширину всех адресных полей пришлось удвоить: 32-битная нумерация фрагментов уступила место 64-битной. Были внесены и другие менее существенные усовершенствования.

Фактически мы имеем дело с тремя различными файловыми системами, не совместимыми друг с другом на уровне базовых структур данных, однако, некоторые источники склонны рассматривать FFS как надстройку над UFS. "UFS (and UFS2) define on-disk data layout. FFS sits on top of UFS (1 or 2) and provides directory structure information, and a variety of disk access optimizations" говорит "Little UFS2 FAQ" (*UFS/UFS2 определяет раскладку данных на диске. FFS реализована поверх UFS 1 или 2 и отвечает за структуру директорий и некоторых дисковых оптимизаций*). Действительно, если заглянуть в исходные тексты файловой системы, можно обнаружить два подкаталога — /ufs и /ffs. В /ffs находится определение суперблока (базовой структуры, отвечающей за раскладку данных), а в /ufs – определение inode и структуры директорий, что опровергает данный тезис, с точки зрения которого все должно быть с точностью до наоборот.

Чтобы не увязнуть в болоте терминологический тонкостей, под UFS мы будем понимать основную файловую систему 4.5 BSD, а под UFS2 – основную файловую систему 5.x BSD.

структура UFS

В начале диске расположен boot-сектор (на не загрузочных разделах он может быть пустым), а все остальное пространство поделено на несколько зон одинакового размера, называемых ***группами цилиндров*** (***cylinder groups***).

Каждая группа цилиндров имеет свою супер-блок (super-block), свою таблицу inod'ов и свою группу блоков данных, совершенно независимую ото всех остальных зон. Другим словами, inod'е описывают блоки данных той и только той зоны, к которой они принадлежат. Это увеличивает быстродействие файловой системы (головка жесткого диска совершает более короткие перемещения) и упрощает процедуру восстановления при значительном разрушении данных, поскольку, как показывает практика, обычно, гибнет только первая группа inod'е. Чтобы погибли все группы… ну я даже не знаю что же такого с жестким диском нужно сделать. А! Знаю! Под пресс положить!

Каждый блок в свою очередь разбит на несколько фрагментов фиксированного размера, предотвращающих потерю свободного пространства в хвостах файлов. Благодаря этому, использование блоков большого размера уже не кажется расточительной идей, напротив, это увеличивает производительность и уменьшает фрагментацию. Если файл использует более одного фрагмента в двух несмежных блоках, он автоматически перемещается на новое место, в наименее фрагментированный регион свободного пространства. Поэтому, фрагментация в UFS очень мала или же совсем отсутствует, что существенно облегчает восстановление удаленных файлов и разрушенных данных.

Адресация ведется либо по физическим смещениям, измеряемых в байтах и отсчитываемых от начала группы цилиндров (реже — UFS-раздела), либо в номерах фрагментов, отсчитываемых от тех же самых точек. Допустим, размер блока составляет 16 Кбайт, разбитых на 8 фрагментов. Тогда 69'й сектор будет иметь смещение 512 x 69 == 35328 байт или 1024 x (16/8)/512 x 69 = 276 фрагментов.

В USF первый суперблок располагается по смещению 8192 байт от конца загрузочного сектора, что соответствует 17-сектору. В UFS2 он "переехал" на 65536 байт (129 секторов) от начала, освобождая место для дисковой метки и первичного загрузчика операционной системы, а для действительно больших (в исходных текстах — piggy, т. е. "свинских") систем предусмотрена возможность перемещения суперблока по адресу 262144 байт (целых 513 секторов)!

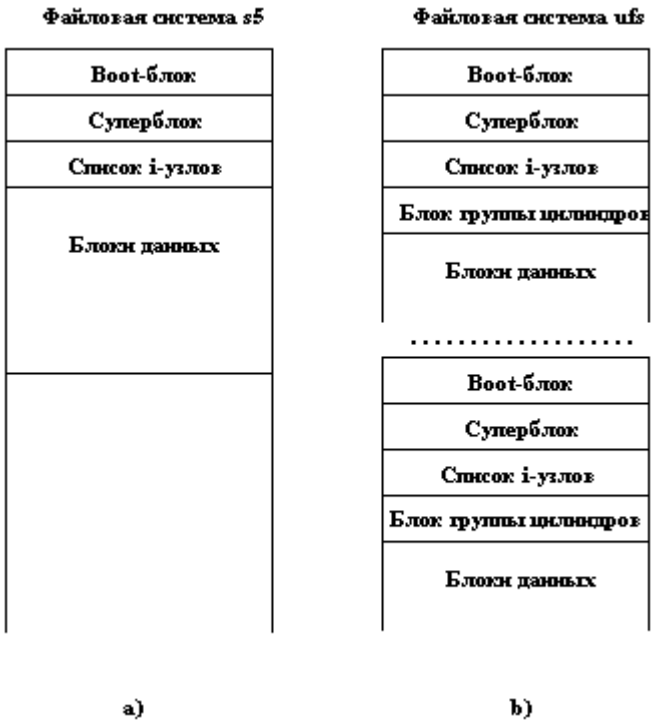

Рисунок 1 структура файловой системы s5/ext2fs (a) и ufs (b)

Вслед за супер-блоком идут одна или несколько групп цилиндров, описываемых ***дескрипторами групп*** (***group descriptors***), ***карты свободного пространства***, в просторечии — *битмапы* (***block bitmap/inode bitmap***) и таблицы ***inode***. Для перестраховки, копия суперблока дублируется в каждой группе. Загрузочный сектор не дублируется, но по соображениям унификации и единообразия, под него просто выделяется место. Таким образом, относительная адресация блоков в каждой группе остается неизменной.

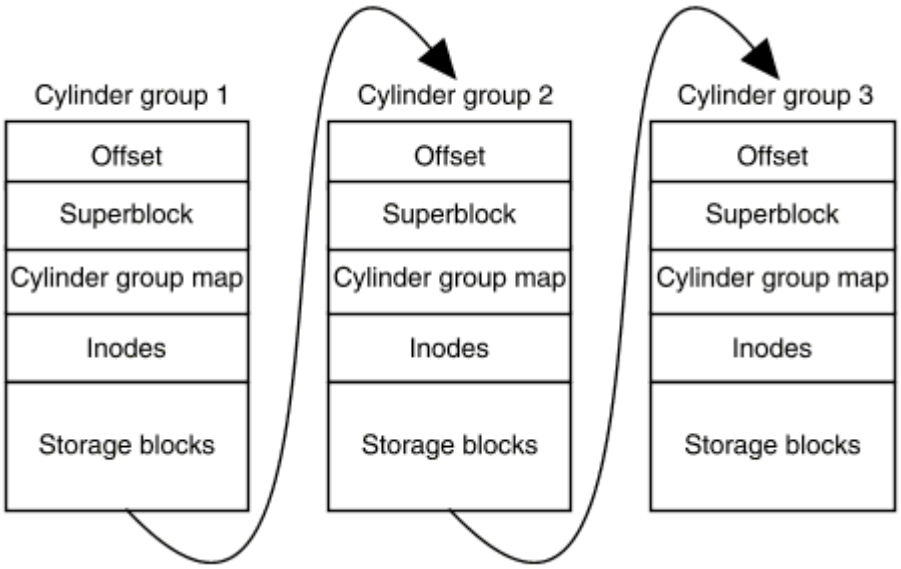

Рисунок 2 последовательно расположенные группы цилиндров

Среди прочей информации суперблок содержит:

- cblkno — смещение первой группы блока цилиндров, измеряемый в фрагментах, отсчитываемых от начала раздела;
- fs_iblkno — смещение первой inode в первой группе цилиндров (фрагменты от начала раздела);

- fs_dblkno — смещение первого блока данных в первой группе цилиндров (фрагменты от начала раздела);
- fs_ncg — кол-во групп цилиндров (штуки);
- fs_bsize – размер одного блока в байтах;
- fs_fsize — размер одного фрагмента в байтах;
- fs_frag — кол-во фрагментов в блоке;
- fs_fpg – размер каждой группы цилиндров, выраженный в блоках (так же может быть найден через fs_cgsize);

Для перевода смещений, выраженных в фрагментах, в номера секторов, служит следующая формула: `sec_n(fragment_offset) = fragment_offset*(fs_bsize/fs_frag/512)` или ее более короткая разновидность: `sec_n(fragment_offset) = fragment_offset*fs_fsize /512`;

Структура суперблока определена в файле /src/ufs/ffs/fs.h и в упрощенном виде выглядит так:

```
struct fs {
/* 0x00 */     int32_t fs_firstfield;         /* historic file system linked list, */
/* 0x04 */     int32_t fs_unused_1;           /*     used for incore super blocks */
/* 0x08 */     ufs_daddr_t fs_sblkno;         /* addr of super-block in filesys */
/* 0x0C */     ufs_daddr_t fs_cblkno;         /* offset of cyl-block in filesys */
/* 0x10 */     ufs_daddr_t fs_iblkno;         /* offset of inode-blocks in filesys */
/* 0x14 */     ufs_daddr_t fs_dblkno;         /* offset of first data after cg */
/* 0x18 */     int32_t fs_cgoffset;           /* cylinder group offset in cylinder */
/* 0x1C */     int32_t fs_cgmask;             /* used to calc mod fs_ntrak */
/* 0x20 */     time_t  fs_time;               /* last time written */
/* 0x24 */     int32_t fs_size;               /* number of blocks in fs */
/* 0x28 */     int32_t fs_dsize;              /* number of data blocks in fs */
/* 0x2C */     int32_t fs_ncg;                /* number of cylinder groups */
/* 0x30 */     int32_t fs_bsize;              /* size of basic blocks in fs */
/* 0x34 */     int32_t fs_fsize;              /* size of frag blocks in fs */
/* 0x38 */     int32_t fs_frag;               /* number of frags in a block in fs */

/* these are configuration parameters */
/* 0x3C */     int32_t fs_minfree;            /* minimum percentage of free blocks */
/* 0x40 */     int32_t fs_rotdelay;           /* num of ms for optimal next block */
/* 0x44 */     int32_t fs_rps;                /* disk revolutions per second */

/* sizes determined by number of cylinder groups and their sizes */
/* 0x98 */     ufs_daddr_t fs_csaddr;         /* blk addr of cyl grp summary area */
/* 0x9C */     int32_t fs_cssize;             /* size of cyl grp summary area */
/* 0xA0 */     int32_t fs_cgsize;             /* cylinder group size */

/* these fields can be computed from the others */
/* 0xB4 */     int32_t fs_cpg;                /* cylinders per group */
/* 0xB8 */     int32_t fs_ipg;                /* inodes per group */
/* 0xBC */     int32_t fs_fpg;                /* blocks per group * fs_frag */

/* these fields are cleared at mount time */
/* 0xD0 */     int8_t  fs_fmod;               /* super block modified flag */
/* 0xD1 */     int8_t  fs_clean;              /* file system is clean flag */
/* 0xD2 */     int8_t  fs_ronly;              /* mounted read-only flag */
/* 0xD3 */     int8_t  fs_flags;              /* see FS_ flags below */
/* 0xD4 */     u_char  fs_fsmnt[MAXMNTLEN];   /* name mounted on */
};
```

Листинг 1 формат супер-блока (второстепенные поля опущены)

За концом супеблока, на некотором отдалении от него, находится первая группа цилиндров. В начале каждой группы расположена служебная структура `cg` — описатель группы цилиндров, содержащая магическую последовательность 55h 02h 09h по которую все уцелевшие группы можно найти даже при полностью испорченном супеблоке (штатным образом, стартовые адреса всех последующих групп вычисляются путем умножения номера группы на ее размер, содержащийся в поле fs_cgsize).

Другие важные параметры:

- cg_cgx — порядковой номер группы, отсчитываемый от нуля;
- cg_old_niblk — кол-во inode в данной группе;
- cg_ndblk — кол-во блоков данных в данной группе;
- csum — кол-во свободных inode и блоков данных в данной группе;

- cg_iusedoff — смещение карты занятых inod'e, отсчитываемое от начала данной группы и измеряемое в байтах;
- cg_freeoff — смещение карты свободного пространства (байты от начла группы);

Структура cg определена в файле /src/ufs/ffs/fs.h и выглядит следующим образом:

```
#define CG_MAGIC        0x090255
#define MAXFRAG         8
struct cg {
/* 0x00 */ int32_t     cg_firstfield;         /* historic cyl groups linked list */
/* 0x04 */ int32_t     cg_magic;              /* magic number */
/* 0x08 */ int32_t     cg_old_time;           /* time last written */
/* 0x0C */ int32_t     cg_cgx;                /* we are the cgx'th cylinder group */
/* 0x10 */ int16_t     cg_old_ncyl;           /* number of cyl's this cg */
/* 0x12 */ int16_t     cg_old_niblk;          /* number of inode blocks this cg */
/* 0x14 */ int32_t     cg_ndblk;              /* number of data blocks this cg */
/* 0x18 */ struct      csum cg_cs;            /* cylinder summary information */
/* 0x28 */ int32_t     cg_rotor;              /* position of last used block */
/* 0x2C */ int32_t     cg_frotor;             /* position of last used frag */
/* 0x30 */ int32_t     cg_irotor;             /* position of last used inode */
/* 0x34 */ int32_t     cg_frsum[MAXFRAG];     /* counts of available frags */
/* 0x54 */ int32_t     cg_old_btotoff;        /* (int32) block totals per cylinder */
/* 0x58 */ int32_t     cg_old_boff;           /* (u_int16) free block positions */
/* 0x5C */ int32_t     cg_iusedoff;           /* (u_int8) used inode map */
/* 0x60 */ int32_t     cg_freeoff;            /* (u_int8) free block map */
/* 0x64 */ int32_t     cg_nextfreeoff;        /* (u_int8) next available space */
/* 0x68 */ int32_t     cg_clustersumoff;      /* (u_int32) counts of avail clusters */
/* 0x6C */ int32_t     cg_clusteroff;         /* (u_int8) free cluster map */
/* 0x70 */ int32_t     cg_nclusterblks;       /* number of clusters this cg */
/* 0x74 */ int32_t     cg_niblk;              /* number of inode blocks this cg */
/* 0x78 */ int32_t     cg_initediblk;         /* last initialized inode */
/* 0x7C */ int32_t     cg_sparecon32[3];      /* reserved for future use */
/* 0x00 */ ufs_time_t  cg_time;               /* time last written */
/* 0x00 */ int64_t     cg_sparecon64[3];      /* reserved for future use */
/* 0x00 */ u_int8_t    cg_space[1];           /* space for cylinder group maps */
/* actually longer */
```

Листинг 2 структура описателя группы цилиндров

Между описателем группы цилиндров и группой inode расположена карта занятых inode и карта свободного дискового пространства, представляющие собой обыкновенные битовые поля, точно такие же как и в NTFS. При восстановлении удаленных файлов без этих карт никуда! Отделяя зерна от плевел, они существенно сужают круг поиска, что особенно хорошо заметно на дисках, заполненных более чем наполовину.

За картами следует массив inod'ов, смещение которого содержится в поле cg_iusedoff (адрес первой группы inode продублирован в суперблоке). В UFS ***inode*** играет ту же самую роль, что и FILE Record в NTFS, в FAT прямых аналогов, к сожалению, нет. Здесь сосредоточена вся информация о файле: тип файла (обычный файл, директория, символьная ссылка и т. д.), логический и физический размер, схема размещения на диске, время создания, модификации, последнего доступа и удаления, правда доступа и количество ссылок на файл.

По сути, в UFS структура inode ничем не отличается от ext2fs, только расположение полей другое. К тому же имеется только один блок косвенной адресации вместо трех, но это уже детали, в которые не будет углубляться (иначе или зависнем или завязнем), а лучше рассмотрим назначение фундаментальных полей, к числу которых принадлежат:

- di_nlink — кол-во ссылок на файл (0 означает "удален");
- di_size — размер файла в байтах;
- di_atime/di_atimensec — время последнего доступа к файлу;
- di_mtime/di_mtimensec — время последней модификации;
- di_ctime/di_ctimensec – время последнего изменения inode;
- di_db – адреса первых 12-блоков данных файла, отсчитываемые в фрагментах от начала группы цилиндров;
- di_ib — адрес блоков косвенной адресации (фрагменты от начала группы);

Сама структура inode определена в файле /src/ufs/ufs/dinode.h и для UFS1 выглядит так:

```
struct dinode {
/* 0x00 */     u_int16_t      di_mode;       /*   0: IFMT, permissions; see below. */
/* 0x02 */     int16_t        di_nlink;      /*   2: File link count. */
```

```
/* 0x04 */     union {
               u_int16_t oldids[2];          /*   4: Ffs: old user and group ids. */
               int32_t inumber;              /*   4: Lfs: inode number. */
        } di_u;
/* 0x08 */     u_int64_t      di_size;       /*   8: File byte count. */
/* 0x10 */     int32_t        di_atime;      /*  16: Last access time. */
/* 0x14 */     int32_t        di_atimensec;  /*  20: Last access time. */
/* 0x18 */     int32_t        di_mtime;      /*  24: Last modified time. */
/* 0x1C */     int32_t        di_mtimensec;  /*  28: Last modified time. */
/* 0x20 */     int32_t        di_ctime;      /*  32: Last inode change time. */
/* 0x24 */     int32_t        di_ctimensec;  /*  36: Last inode change time. */
/* 0x28 */     ufs_daddr_t    di_db[NDADDR]; /*  40: Direct disk blocks. */
/* 0x58 */     ufs_daddr_t    di_ib[NIADDR]; /*  88: Indirect disk blocks. */
/* 0x64 */     u_int32_t      di_flags;      /* 100: Status flags (chflags). */
/* 0x68 */     int32_t        di_blocks;     /* 104: Blocks actually held. */
/* 0x6C */     int32_t        di_gen;        /* 108: Generation number. */
/* 0x70 */     u_int32_t      di_uid;        /* 112: File owner. */
/* 0x74 */     u_int32_t      di_gid;        /* 116: File group. */
/* 0x78 */     int32_t        di_spare[2];   /* 120: Reserved; currently unused */
};
```

Листинг 3 структура inode в USF1

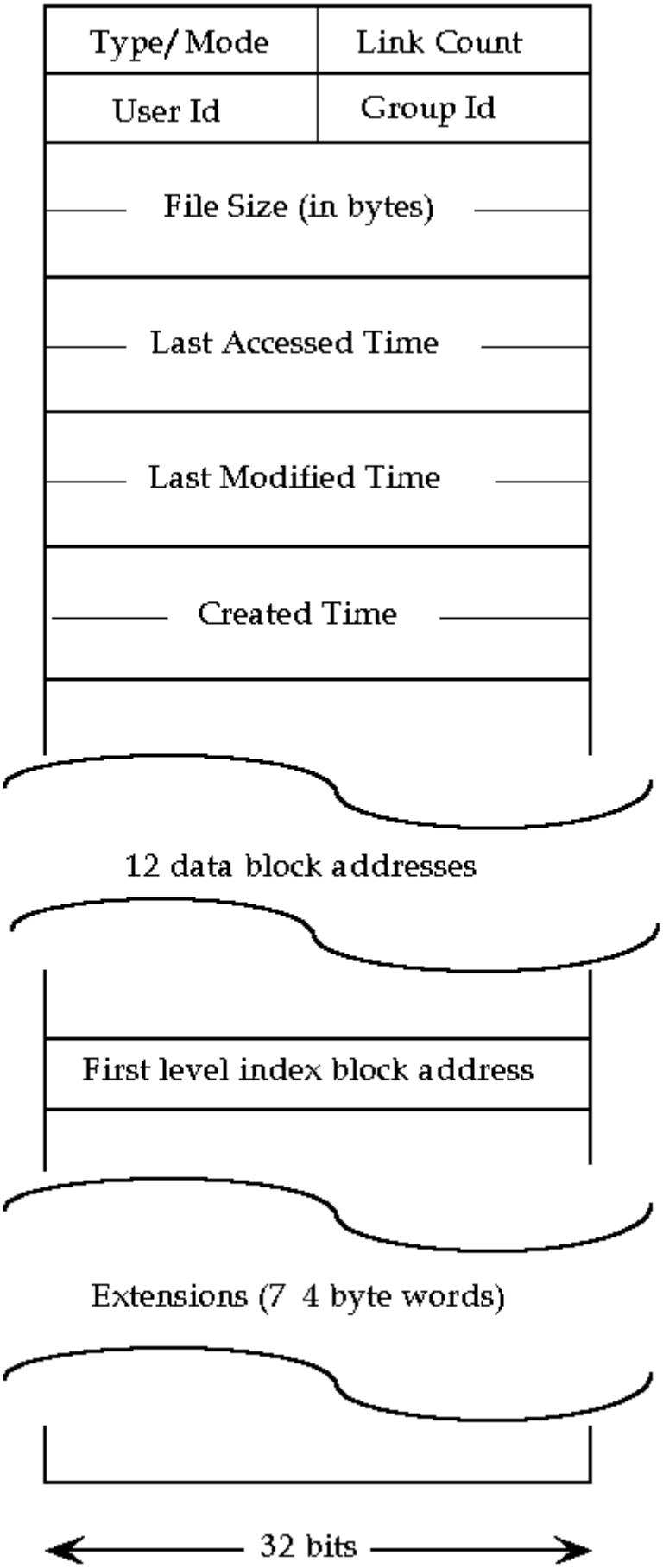
Type/Mode
Link Count
User Id
Group Id
File Size (in bytes)
Last Accessed Time
Last Modified Time
Created Time
12 data block addresses
First level index block address
Extensions (7 4 byte words)
32 bits

Рисунок 3 схематичное изображение inode

В UFS2 формат inode был существенно изменен — появилось множество новых полей, удвоилась ширина адресных полей и т. д. Что это обозначает для нас в практическом плане? Смещения всех полей изменились, только и всего, а общий принцип работы с inod'ами остался прежним:

```
struct ufs2_dinode {
/* 0x00 */ u_int16_t  di_mode;       /*   0: IFMT, permissions; see below. */
/* 0x02 */ int16_t    di_nlink;      /*   2: File link count. */
/* 0x04 */ u_int32_t  di_uid;        /*   4: File owner. */
/* 0x08 */ u_int32_t  di_gid;        /*   8: File group. */
/* 0x0C */ u_int32_t  di_blksize;    /*  12: Inode blocksize. */
/* 0x10 */ u_int64_t  di_size;       /*  16: File byte count. */
/* 0x18 */ u_int64_t  di_blocks;     /*  24: Bytes actually held. */
/* 0x20 */ ufs_time_t di_atime;      /*  32: Last access time. */
/* 0x28 */ ufs_time_t di_mtime;      /*  40: Last modified time. */
/* 0x30 */ ufs_time_t di_ctime;      /*  48: Last inode change time. */
/* 0x38 */ ufs_time_t di_birthtime;  /*  56: Inode creation time. */
/* 0x40 */ int32_t    di_mtimensec;  /*  64: Last modified time. */
/* 0x44 */ int32_t    di_atimensec;  /*  68: Last access time. */
/* 0x48 */ int32_t    di_ctimensec;  /*  72: Last inode change time. */
/* 0x4C */ int32_t    di_birthnsec;  /*  76: Inode creation time. */
/* 0x50 */ int32_t    di_gen;        /*  80: Generation number. */
/* 0x54 */ u_int32_t  di_kernflags;  /*  84: Kernel flags. */
/* 0x58 */ u_int32_t  di_flags;      /*  88: Status flags (chflags). */
/* 0x5C */ int32_t    di_extsize;    /*  92: External attributes block. */
/* 0x60 */ ufs2_daddr_tdi_extb[NXADDR];/*  96: External attributes block. */
/* 0x70 */ ufs2_daddr_tdi_db[NDADDR]; /* 112: Direct disk blocks. */
/* 0xD0 */ ufs2_daddr_tdi_ib[NIADDR]; /* 208: Indirect disk blocks. */
/* 0xE8 */ int64_t    di_spare[3];   /* 232: Reserved; currently unused */
 };
```

Листинг 4 структура inode в USF2

Имена файлов хранятся в директориях. В inod'ах их нет. С точки зрения UFS, директории являются обыкновенными файлами (ну, может, не совсем обыкновенными) и могут хранится в любом месте, принадлежащем группе цилиндров. Файловая система UFS поддерживает несколько типов хеширования директорий, однако на структуре хранения имен это никак не отражается. Имена хранятся в блоках, называемых DIRBLKSIZ в структурах типа direct, выровненных по 4'х байтовой границе.

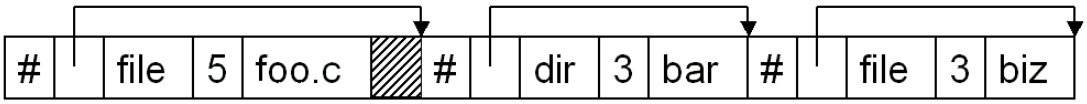

Рисунок 4 хранение имен файлов и директорий

Структура direct определена в файле /src/ufs/ufs/dir.h и содержит: номер inode, описывающий данный файл, тип файла, его имя, а так же длину самой структуры direct, используемую для нахождения следующего direct'а в блоке.

```
struct  direct {
/* 0x00 */ u_int32_t  d_ino;                /* inode number of entry */
/* 0x04 */ u_int16_t  d_reclen;             /* length of this record */
/* 0x06 */ u_int8_t   d_type;               /* file type, see below */
/* 0x07 */ u_int8_t   d_namlen;             /* length of string in d_name */
/* 0x08 */ char       d_name[MAXNAMLEN + 1];/* name with length <= MAXNAMLEN */
};
```

Листинг 5 структура direct, отвечающая за хранение имен файлов и директорий

На этом описание файловой системы UFS можно считать законченным. Для ручного восстановления данных приведенной информации вполне достаточно.

что происходит при удалении файла

При удалении файла на UFS-разделе происходит следующее (события перечислены в порядке расположения соответствующих структур в разделе и могут не совпадать с порядком их возникновения):

- в суперблоке обновляется поле fs_time (время последнего доступа к разделу);
- в суперблоке обновляется структура fs_cstotal (кол-во свободных inod'ов и блоков данных в разделе);
- в группе цилиндров обновляются карты занятых inod'ов и блоков данных — inod'е и все блоки данных удаляемого файла помечаются как освобожденные;
- в indoe материнского каталога обновляются поля времени последнего доступа и модификации;
- в indoe материнского каталога обновляется поле времени последнего изменения inode;
- в inode удаляемого файла поля di_mode (IFMT, permissions), di_nlink (кол-во ссылок на файл) и di_size (размер файла) варварски обнуляются;
- в inode удаляемого файла поля di_db (массив указателей на 12 первых блоков файла) и di_ib (указатель на блок косвенной адресации) безжалостно затираются нулями;
- в inode удаляемого файла обновляются поля времени последней модификации и изменения inod'е, время последнего доступа при этом остается неизменным;
- в inode удаляемого файла обновляется поле di_spare. В исходных текстах оно помечено как "Reserved; currently unused", но просмотр дампа показывает, что это не так. Судя по всему здесь хранится нечто вроде последовательности обновления (update sequence), используемой для контроля целостности indoe, однако, это только предположение;
- в директории удаленного файла, размер предшествующей структуры direct увеличивается на d_reclen, в результате чего она как бы "поглощает" имя удаляемого файла, однако, его затирания не происходит, во всяком случае оно затирается не сразу, а только тогда, когда в этом возникнет реальная необходимость;

подготовка к восстановлению

Если вы только что удалили файл, то лучшим способом восстановления будет RESET. Без шуток! Система сбрасывает дисковые буфера не сразу, а спустя некоторое время, поэтому своевременная перезагрузка или отключение питания часто спасает ситуацию и после загрузки файл окажется цел и невредим, правда, на самом диске могут образоваться значительные разрушения, так что риск неблагоприятного исхода очень велик и лучше воспользоваться более традиционными средствами восстановления.

Первым делом размонтируете (unmount) дисковый раздел или перемонтируете его "только на чтение". Лечение активных разделов обычно заканчивается очень печально. Если восстанавливаемые файлы находятся на системном разделе, в этом случае можно прибегнуть к LiveCD. Лучше всего использовать KNOPPIX. Он поддерживает большое количество оборудования, не требователен к ресурсам (достаточно всего 128 Мбайт памяти) и содержит все необходимые утилиты для восстановления. Опытные пользователи могут сформировать загрузочный CD или даже дискету самостоятельно.

Широко разрекламированный дистрибутив **Ferenzy 0.3**, основанный на Free BSD лучше сразу выкинуть на помойку — в нем совсем немного дисковых утилит, да и те ориентированны в основном на ext2fs, и USF/FFS они поддерживает постольку-поскольку. Тем не менее для восстановительных работ данный диск вполне пригоден, и если ничего другого под рукой нет…

Дисковых редакторов, работающий на секторов уровне, под BSD существует не так уж и много. Обычно для этой цели пользуются BSD-портом LINUX-редактора lde (http://lde.sourceforge.net/). К сожалению, на моем системе (4.5 BSD) он работает крайне нестабильно и не отображает основные структуры данных в удобочитаемом виде, хотя поддержка UFS в нем заявлена. В принципе, можно вставить в привод загрузочный CD-ROM с Windows PE и воспользоваться любым Windows-редактором от Microsoft Disk Probe до Runtime Disk Explorer'a. То же самое справедливо и для Norton Disk Editor'a, запущенного с дискеты из-под MS-DOS (правда ни диски большого объема, ни SCSI-устройства он не поддерживает). Еще можно запустить KNOPPIX или любой Live LINUX, ориентированный на восстановление, но дело в том, что редактируя диск напрямую, его легко испортить. Одно неверное движение руки — и гигабайты данных обращаются в прах.

При наличии свободного места, рекомендуется создать копию раздела и все дальнейшие опыты проводить уже над ней. В мире Windows для этой цели требуется

специальные утилиты (например, Norton Ghost), которые, кстати говоря, стоят нехилых денег, но BDS – совсем другое дело. Здесь все необходимое находится под рукой. Копию раздела проще всего создать командой `cp /dev/ad0s1a dump`, где ad0s1a – имя устройства, а dump — имя файла-дампа, для работы с которым сгодиться любой hex-редактор, например, biew (http://biew.sourceforge.net/).

В общем, возможностей много, но как говорится, на вкус и цвет товарищей нет…

техника ручного восстановления файлов

Начнем с грустного. Поскольку, при удалении файла ссылки на 12 первых блоков и 3 блока косвенной адресации необратимо затираются, автоматическое восстановление данных невозможно в принципе. Найти удаленный файл можно только по его содержимому. Искать, естественно, необходимо в свободном пространстве. Вот тут-то нам и пригодятся карты, расположенные за концом описателя группы цилиндров.

Если нам повезет и файл окажется нефрагментированным (а на UFS, как уже отмечалось, фрагментация обычно отсутствует или крайне невелика), остальное будет делом техники. Просто выделяем группу секторов и записываем ее на диск, но только ни в коем случае не на сам восстанавливаемый раздел! (Например, файл можно передать на соседнюю машину по сети). К сожалению, поле длины файла безжалостно затирается при его удалении и актуальный размер приходится определять "на глазок". Звучит намного страшнее, чем выглядит. Неиспользуемый хвост последнего фрагмента всегда забивается нулями, что дает хороший ориентир. Проблема в том, что некоторые типы файлов содержат в своем конце некоторое количество нулей, при отсечении которых их работоспособность нарушается, поэтому тут приходится экспериментировать.

А если файл фрагментирован? Первые 13 блоков (именно блоков, а не фрагментов!) придется собирать руками. В идеале это будет один непрерывный регион. Хуже, если первый фрагмент расположен в "чужом" блоке (т. е. блоке частично занятом другим файлом), а оставшиеся 12 блоков находятся в одном или нескольких регионах. Вообще-то достаточно трудно представить себе ситуацию, в которой первые 13 блоков были бы сильно фрагментированы (а поддержка фоновой дефрагментации в UFS на что?) Такое может произойти только при интересной "перегруппировке" большого количеств файлов, что в реальной жизни практически никогда не встречается (ну разве только что вы задумали навести порядок на своем жестком диске). Короче, будем считать, что 13'й блок файла найден. В массив непосредственной адресации он уже не влезает (там содержатся только 12 блоков) и ссылка на него, как и на все последующие блоки файла, должна содержаться в блоках косвенной адресации, которые при удалении файла помечается как свободные но не затирается, точнее затираются, но не сразу. Большинство файлов обходятся только одним косвенным блоком, что существенно упрощает нашу задачу.

Как найти этот блок на диске? Вычисляем смещение 13'го блока файла от начала группы цилиндров, переводим его в фрагменты, записываем получившееся число задом наперед (так, чтобы младшие байты располагались по меньшим адресами) и осуществляем контекстный поиск в свободном пространстве.

Отличить блок косвенной адресации от всех остальных типов данных очень легко — он представляет собой массив указателей на блоки, а в конце идут нули. Остается только извлечь эти блоки с диска и записать их в файл, обрезая его по нужной длине. Внимание! Если вы нашли несколько "кандидатов" в блоки косвенной адресации, это означает, что 13'й блок удаленного файла в разное время принадлежал различным файлам (а так, скорее всего и будет). Не все косвенные блоки были затерты, вот ссылки и остались. Как отличить "наш" блок от "чужих"? Если хотя бы одна из ссылок указывает на уже занятый блок данных (что легко определить по карте), такой блок можно сразу откинуть. Оставшиеся блоки перебираются вручную до получения работоспособной копии файла. Имя файла (если оно еще не затерто) можно извлечь из директории. Естественно, при восстановлении нескольких файлов мы не можем однозначно сказать какое из имен какому файлу принадлежит, тем не менее это все же лучше, чем совсем ничего. Директории восстанавливаются точно так же как и обыкновенные файлы, хотя по правде говоря, в них кроме имен файлов нечего восстанавливать…

заключение

Описанный метод восстановления данных страдает кучей ограничений. В частности, при удалении большого количества сильно фрагментированных двоичных файлов он говорит

"пас" и уходит в кусты. Вы только убьете свое время, но навряд ли найдете среди обломков файловой системы что-то полезное. Но как бы там ни было, другого выхода просто нет (если, конечно, не считать резервной копию, которой тоже нет), поэтому, мыщъх все-таки считает, что данная статья будет совсем небесполезной.