# ВОССТАНАВЛИВАЕМ УДАЛЕННЫЕ ФАЙЛЫ ПОД BSD

*Командной строке посвящается…*

**UFS – это основная файловая система для BSD-систем, устанавливаемая по умолчанию. Многие коммерческие UNIX также используют либо саму UFS, либо нечто очень на нее похожее. В противоположность ext2fs, исхоженной вдоль и поперек, UFS крайне поверхностно описана в доступной литературе и 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 поверку все они оказываются неработоспособными. Эта статья описывает основные структуры файловой системы и рассказывает как восстанавливать данные вручную.**

**КРИС КАСПЕРСКИ**

## Немного истории

UFS 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, средняя производительность FS составляла всего лишь 2%-5% от «сырой» производительности жесткого 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, S5 FS претерпела некоторые улучшения. Были добавлены длинные имена, символические ссылки и т. д. Так родилась UFS.

В 4.3 BSD, увидевшей свет уже в 1984 году, улучшения носили намного более радикальный, если не сказать революционный, характер. Появились концепции фрагментов (fragments) и групп цилиндров (cylinder groups). Быстродействие файловой системы 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 некоторые оптимизации доступа к диску). Если заглянуть в исходные тексты файловой системы, действительно, 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 позиций которого все должно быть с точностью до наоборот.

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

## Структура UFS

Внешне UFS очень похожа на ext2fs – те же inode, блоки данных, файлы, директории… Но есть и отличия. В ext2fs имеется только одна группа inode и только одна группа блоков данных для всего раздела. UFS же делит раздел на несколько зон одинакового размера, называемых группами цилиндров. Каждая зона имеет свою группу inode и свою группу блоков данных, независимую от всех остальных зон. Другим словами, inode 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 inode. Чтобы погибли все группы… я даже не знаю, что же такое с жестким диском нужно сделать. А! Знаю! Под пресс положить!

В UFS каждый блок разбит на несколько фрагментов фиксированного размера, предотвращающих потерю свободного пространства в хвостах файлов. Благодаря этому, использование блоков большого размера уже не кажется расточительной идеей, 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.

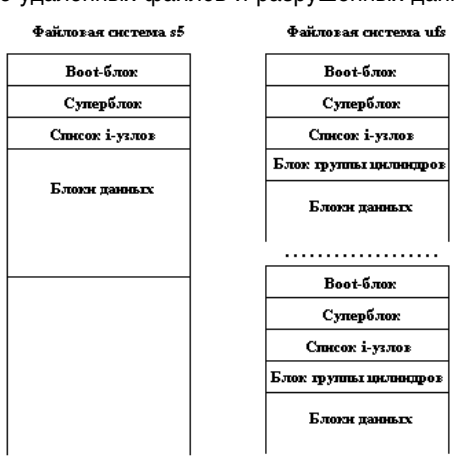


Рисунок 1. Структура файловой системы s5/ext2fs (a) и ufs (b)

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 Кб, 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 начале раздела расположен загрузочный сектор, затем следует суперблок, за которым находится одна или несколько групп цилиндров. 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.

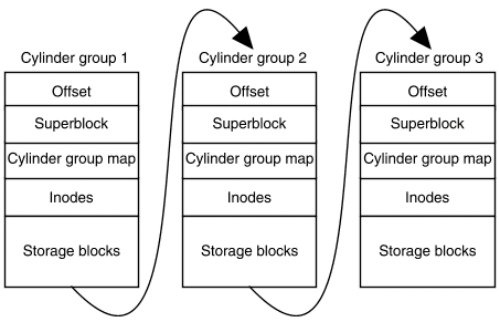


Рисунок 2. Последовательно расположенные группы цилиндров

В UFS суперблок располагается по смещению 8192 байт от начала раздела, что соответствует 16-му сектору. В UFS2 он «переехал» на 65536 байт (128 секторов) от начала, освобождая место для дисковой метки и первичного загрузчика операционной системы, а для действительно больших (в исходных текстах – piggy) систем предусмотрена возможность перемещения суперблока по адресу 262144 байт (целых 512 секторов)!

Среди прочей информации суперблок содержит:

- fs_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 – количество групп цилиндров (штуки);
- fs_bsize – размер одного блока в байтах;
- fs_fsize – размер одного фрагмента в байтах;
- fs_frag – количество фрагментов в блоке;
- fs_fpg – размер каждой группы цилиндров, выраженный в блоках (также может быть найден через fs_cgsize).

Для перевода смещений, выраженных в фрагментах, в номера секторов используется 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:

Листинг 1. Формат суперблока (второстепенные поля опущены)

```
struct fs {
/* historic file system linked list, */
```

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

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

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

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

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

За суперблоком находится первая группа цилиндров. В начале каждой группы расположена служебная структура cg (далее по тексту – описатель группы цилиндров, термин автора), содержащая магическую последовательность 55h 02h 09h, позволяющая находить уцелевшие группы цилиндров даже при полном разрушении суперблока (если же суперблок цел, стартовые адреса всех последующих групп вычисляются путем умножения номера группы на ее размер, содержащийся в поле fs_cgsize).

Другие важные параметры описателя группы цилиндров:

- cg_cgx – порядковой номер группы, отсчитываемый от нуля;
- cg_old_niblk – количество inode в данной группе;
- cg_ndblk – количество блоков данных в данной группе;
- csum – количество свободных inode и блоков данных в данной группе;
- cg_iusedoff – смещение карты занятых inode, отсчитываемое от начала данной группы и измеряемое в байтах;
- cg_freeoff – смещение карты свободного пространства (байты от начала группы).

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

Листинг 2. Структура описателя группы цилиндров

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

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 в NTFS или ext2fs/ext3fs. 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 inode, смещение которого содержится в поле cg_iusedoff (адрес первой группы inode продублирован в суперблоке). По сути, в UFS структура inode ничем не отличается от ext2fs, только расположение полей другое.

Давайте лучше рассмотрим назначение основных полей inode, к числу которых принадлежат:

- di_nlink – количество 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 (см. **листинг 3** и **рис. 3**):

Листинг 3. Структура inode в USF1

```
struct dinode {
/*    0: IFMT, permissions; see below. */
/* 0x00 */ u_int16_t      di_mode;
/*    2: File link count. */
/* 0x02 */ int16_t        di_nlink;
/* 0x04 */ union {
/*    4: Ffs: old user and group ids. */
           u_int16_t      oldids[2];
/*    4: Lfs: inode number. */
           int32_t        inumber;
    } di_u;
/*    8: File byte count. */
/* 0x08 */ u_int64_t      di_size;
/*   16: Last access time. */
/* 0x10 */ int32_t        di_atime;
/*   20: Last access time. */
/* 0x14 */ int32_t        di_atimensec;
/*   24: Last modified time. */
/* 0x18 */ int32_t        di_mtime;
/*   28: Last modified time. */
/* 0x1C */ int32_t        di_mtimensec;
/*   32: Last inode change time. */
/* 0x20 */ int32_t        di_ctime;
/*   36: Last inode change time. */
/* 0x24 */ int32_t        di_ctimensec;
/*   40: Direct disk blocks. */
/* 0x28 */ ufs_daddr_t    di_db[NDADDR];
/*   88: Indirect disk blocks. */
/* 0x58 */ ufs_daddr_t    di_ib[NIADDR];
/* 100: Status flags (chflags). */
/* 0x64 */ u_int32_t      di_flags;
/* 104: Blocks actually held. */
/* 0x68 */ int32_t        di_blocks;
/* 108: Generation number. */
/* 0x6C */ int32_t        di_gen;
/* 112: File owner. */
/* 0x70 */ u_int32_t      di_uid;
/* 116: File group. */
/* 0x74 */ u_int32_t      di_gid;
/* 120: Reserved; currently unused */
/* 0x78 */ int32_t        di_spare[2];
};
```

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 inode остался прежним (см. **листинг 4**).

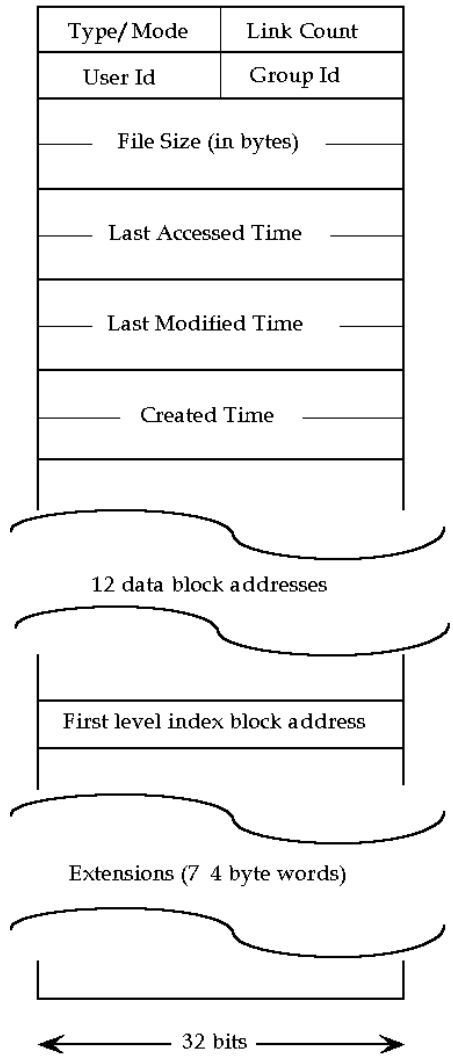


Рисунок 3. Схематичное изображение inode. К полю Extensions относится все, что находится ниже di_ib

Листинг 4. Структура inode в USF2

```
struct ufs2_dinode {
/*    0: IFMT, permissions; see below. */
/* 0x00 */ u_int16_t      di_mode;
/*    2: File link count. */
/* 0x02 */ int16_t        di_nlink;
/*    4: File owner. */
/* 0x04 */ u_int32_t      di_uid;
/*    8: File group. */
/* 0x08 */ u_int32_t      di_gid;
/*   12: Inode blocksize. */
/* 0x0C */ u_int32_t      di_blksize;
/*   16: File byte count. */
/* 0x10 */ u_int64_t      di_size;
/*   24: Bytes actually held. */
/* 0x18 */ u_int64_t      di_blocks;
/*   32: Last access time. */
/* 0x20 */ ufs_time_t     di_atime;
/*   40: Last modified time. */
/* 0x28 */ ufs_time_t     di_mtime;
/*   48: Last inode change time. */
/* 0x30 */ ufs_time_t     di_ctime;
/*   56: Inode creation time. */
/* 0x38 */ ufs_time_t     di_birthtime;
/*   64: Last modified time. */
/* 0x40 */ int32_t        di_mtimensec;
/*   68: Last access time. */
/* 0x44 */ int32_t        di_atimensec;
/*   72: Last inode change time. */
/* 0x48 */ int32_t        di_ctimensec;
/*   76: Inode creation time. */
/* 0x4C */ int32_t        di_birthnsec;
/*   80: Generation number. */
/* 0x50 */ int32_t        di_gen;
```

```
/*  84: Kernel flags. */
/* 0x54 */ u_int32_t     di_kernflags;
/*  88: Status flags (chflags). */
/* 0x58 */ u_int32_t     di_flags;
/*  92: External attributes block. */
/* 0x5C */ int32_t       di_extsize;
/*  96: External attributes block. */
/* 0x60 */ ufs2_daddr_tdi_extb[NXADDR];
/* 112: Direct disk blocks. */
/* 0x70 */ ufs2_daddr_tdi_db[NDADDR];
/* 208: Indirect disk blocks. */
/* 0xD0 */ ufs2_daddr_tdi_ib[NIADDR];
/* 232: Reserved; currently unused */
/* 0xE8 */ int64_t       di_spare[3];
 };
```

Имена файлов хранятся в директориях. В inode их нет. С точки зрения UFS, директории являются обыкновенными файлами и могут храниться 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 DIRECT BLOCK в структурах типа direct, выровненных по 4-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 также длину самой структуры direct, используемую для нахождения следующего direct в блоке.

Листинг 5. Структура direct, отвечающая за хранение имен файлов и директорий

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

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

## На обломках империи

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 (количество свободных inode и блоков данных в разделе);
- в группе цилиндров обновляются карты занятых inode и блоков данных – inode, и все блоки данных удаляемого файла помечаются как освобожденные;
- в inode материнского каталога обновляются поля времени последнего доступа и модификации;
- в inode материнского каталога обновляется поле времени последнего изменения inode;
- в inode удаляемого файла поля di_mode (IFMT – Input Format – формат файла, permissions – права доступа), di_nlink (количество ссылок на файл) и di_size (размер файла) варварски обнуляются (замечание: если счетчик ссылок был больше единицы, то при удалении одной из таких ссылок поле di_nlink уменьшается на единицу, но удаления самого файла не происходит);
- в inode удаляемого файла поля di_db (массив указателей на 12 первых блоков файла), и di_ib (указатель на блок косвенной адресации) безжалостно обнуляется;
- в inode удаляемого файла обновляются поля времени последней модификации и изменения inode, 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. Здесь хранится нечто вроде последовательности обновления (update sequence), используемой для контроля целостности indoe, однако это только предположение;

## Чем восстанавливать?

1. Компания Stellarinfo (www.stellarinfo.com) выпустила утилиту «Phoenix», предназначенную для восстановления данных и поддерживающую практически все популярные файловые системы, которые только известны на сегодняшний день (и UFS в том числе). Демонстрационную копию можно скачать по адресу: http://www.stellarinfo.com/spb.exe. Обратите внимание на расширение файла. Это «.exe». Судя по графе «Platform Supported», он рассчитан на BSD. Но в BSD он не запустится! Потребуется устанавливать в систему дополнительный винчестер с рабочей Windows и инсталлировать Phoenix поверх нее. Под Windows PE он работать отказывается… На Windows 2000 запускается, но при попытке анализа заведомо исправного раздела он аварийно завершается с сообщением о критической ошибке. На других системах я его не проверял. Тем не менее ссылку на файл все-таки даю. Во-первых, вы будете знать, что это за продукт, а во-вторых, не исключено, что у кого-то он все-таки сработает.
2. The Sleuth Kit представляет собой бесплатно распространяемый комплект утилит для ручного восстановления файловой системы, который можно найти по адресу (http://www.sleuthkit.org), там же (http://www.sleuthkit.org/sleuthkit/docs/ref_fs.html) находится краткий how-to. Увы, чудес не бывает, и вся методика восстановления сводится к сканированию свободного пространства на предмет поиска фрагментов с известным содержимым.
3. Foremost – еще одна бесплатная утилита для восстановления удаленных файлов, основанная формате их заголовков на особенностях структуры. Естественно, она работает только с теми файлами, чье строение ей известно. Тем не менее, по сравнению с ее предшественницами это большой шаг вперед! Кстати говоря, утилита взаимодействует с файловой системой не напрямую, а обрабатывает файлы, полученные командой dd или набором Sleuth Kit, благодаря чему она «поддерживает» все файловые системы. Последняя версия лежит на сервере http://foremost.sourceforge.net.

- в директории удаленного файла размер предшествующей структуры direct увеличивается на d_reclen, в результате чего она как бы «поглощает» имя удаляемого файла, однако его перезаписывания не происходит, во всяком случае оно уничтожается не сразу, а только тогда, когда в этом возникнет реальная необходимость.

## Как мы будем действовать

После непреднамеренного удаления одного или нескольких файлов немедленно демонтируйте раздел и запустите дисковый редактор, работающий на уровне секторов. Например, можно воспользоваться BSD-портом уже известной нам утилитой lde. К сожалению, на моей системе (4.5 BSD) она работает крайне нестабильно и не отображает основных структур данных в удобочитаемом виде, хотя поддержка UFS в ней заявлена. При наличии достаточного количества свободного места можно скопировать раздел в файл и открыть его с помощью любого hex-редактора (например, biew) или обратиться непосредственно к самому устройству раздела (например, /dev/ad0s1a).

А еще можно вставить в привод загрузочный CD-ROM с Windows PE и воспользоваться любым Windows-редактором от Microsoft Disk Probe до Runtime Disk Explorer. То же самое справедливо и для Norton Disk Editor, 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 (правда, в большинстве «реанимационных» дистрибутивов, и в частности, Frenzy 0.3, никакого дискового редактора вообще нет!).

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

## Техника 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). Итак, будем считать, что 13-й блок файла найден. В массив непосредственной адресации он уже не помещается (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 помечаются как свободные, но не перезаписывается, точнее, перезаписывается, 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 и восстанавливать…

## Заключение

Описанный метод восстановления данных страдает множеством ограничений. В частности, при удалении большого количества сильно фрагментированных двоичных файлов он ничем не поможет. 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 копии, которой тоже, скорее всего, нет), поэтому я все-таки считаю, что данная статья будет совсем небесполезной.