NAME

fs, **inode** - format of file system volume

SYNOPSIS

#include <sys/param.h>
#include <ufs/ffs/fs.h>

#include <sys/types.h>
#include <sys/lock.h>
#include <sys/extattr.h>
#include <sys/acl.h>
#include <ufs/ufs/quota.h>
#include <ufs/ufs/dinode.h>
#include <ufs/ufs/extattr.h>

DESCRIPTION

The files <*fs.h*> and <*inode.h*> declare several structures, defined variables and macros which are used to create and manage the underlying format of file system objects on random access devices (disks).

The block size and number of blocks which comprise a file system are parameters of the file system. Sectors beginning at BBLOCK and continuing for BBSIZE are used for a disklabel and for some hardware primary and secondary bootstrapping programs.

The actual file system begins at sector SBLOCK with the *super-block* that is of size SBLOCKSIZE. The following structure describes the super-block and is from the file <*ufs/ffs/fs.h*>:

```
/*
 * Super block for an FFS filesystem.
 */
struct fs {
        int32_t     fs_firstfield;          /* historic filesystem linked list, */
        int32_t     fs_unused_1;    /*     used for incore super blocks */
        int32_t     fs_sblkno;      /* offset of super-block in filesys */
        int32_t     fs_cblkno;      /* offset of cyl-block in filesys */
        int32_t     fs_iblkno;      /* offset of inode-blocks in filesys */
        int32_t     fs_dblkno;      /* offset of first data after cg */
        int32_t     fs_old_cgoffset;  /* cylinder group offset in cylinder */
        int32_t     fs_old_cgmask;    /* used to calc mod fs_ntrak */
        int32_t  fs_old_time;     /* last time written */
        int32_t     fs_old_size;      /* number of blocks in fs */
```

```
        int32_t     fs_old_dsize;     /* number of data blocks in fs */
        int32_t     fs_ncg;           /* number of cylinder groups */
        int32_t     fs_bsize;         /* size of basic blocks in fs */
        int32_t     fs_fsize;         /* size of frag blocks in fs */
        int32_t     fs_frag;          /* number of frags in a block in fs */
/* these are configuration parameters */
        int32_t     fs_minfree;       /* minimum percentage of free blocks */
        int32_t     fs_old_rotdelay;  /* num of ms for optimal next block */
        int32_t     fs_old_rps;       /* disk revolutions per second */
/* these fields can be computed from the others */
        int32_t     fs_bmask;         /* ''blkoff'' calc of blk offsets */
        int32_t     fs_fmask;         /* ''fragoff'' calc of frag offsets */
        int32_t     fs_bshift;        /* ''lblkno'' calc of logical blkno */
        int32_t     fs_fshift;        /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
        int32_t     fs_maxcontig;     /* max number of contiguous blks */
        int32_t     fs_maxbpg;        /* max number of blks per cyl group */
/* these fields can be computed from the others */
        int32_t     fs_fragshift;     /* block to frag shift */
        int32_t     fs_fsbtodb;       /* fsbtodb and dbtofsb shift constant */
        int32_t     fs_sbsize;        /* actual size of super block */
        int32_t     fs_spare1[2];     /* old fs_csmask */
                                      /* old fs_csshift */
        int32_t     fs_nindir;        /* value of NINDIR */
        int32_t     fs_inopb;         /* value of INOPB */
        int32_t     fs_old_nspf;      /* value of NSPF */
/* yet another configuration parameter */
        int32_t     fs_optim;         /* optimization preference, see below */
        int32_t     fs_old_npsect;    /* # sectors/track including spares */
        int32_t     fs_old_interleave; /* hardware sector interleave */
        int32_t     fs_old_trackskew; /* sector 0 skew, per track */
        int32_t     fs_id[2];         /* unique filesystem id */
/* sizes determined by number of cylinder groups and their sizes */
        int32_t     fs_old_csaddr;          /* blk addr of cyl grp summary area */
        int32_t     fs_cssize;        /* size of cyl grp summary area */
        int32_t     fs_cgsize;        /* cylinder group size */
        int32_t     fs_spare2;        /* old fs_ntrak */
        int32_t     fs_old_nsect;     /* sectors per track */
        int32_t  fs_old_spc;       /* sectors per cylinder */
        int32_t     fs_old_ncyl;      /* cylinders in filesystem */
```

```
        int32_t     fs_old_cpg;       /* cylinders per group */
        int32_t     fs_ipg;           /* inodes per group */
        int32_t     fs_fpg;           /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
        struct      csum fs_old_cstotal; /* cylinder summary information */
/* these fields are cleared at mount time */
        int8_t   fs_fmod;         /* super block modified flag */
        int8_t   fs_clean;        /* filesystem is clean flag */
        int8_t      fs_ronly;         /* mounted read-only flag */
        int8_t   fs_old_flags;    /* old FS_ flags */
        u_char      fs_fsmnt[MAXMNTLEN]; /* name mounted on */
        u_char      fs_volname[MAXVOLLEN]; /* volume name */
        uint64_t fs_swuid;        /* system-wide uid */
        int32_t  fs_pad;          /* due to alignment of fs_swuid */
/* these fields retain the current block allocation info */
        int32_t     fs_cgrotor;       /* last cg searched */
        void        *fs_ocsp[NOCSPTRS]; /* padding; was list of fs_cs buffers */
        uint8_t *fs_contigdirs;   /* # of contiguously allocated dirs */
        struct      csum *fs_csp;     /* cg summary info buffer for fs_cs */
        int32_t     *fs_maxcluster;   /* max cluster in each cyl group */
        u_int       *fs_active;       /* used by snapshots to track fs */
        int32_t     fs_old_cpc;       /* cyl per cycle in postbl */
        int32_t     fs_maxbsize;      /* maximum blocking factor permitted */
        int64_t     fs_unrefs;        /* number of unreferenced inodes */
        int64_t     fs_sparecon64[16]; /* old rotation block list head */
        int64_t     fs_sblockloc;     /* byte offset of standard superblock */
        struct      csum_total fs_cstotal;  /* cylinder summary information */
        ufs_time_t fs_time;       /* last time written */
        int64_t     fs_size;          /* number of blocks in fs */
        int64_t     fs_dsize;         /* number of data blocks in fs */
        ufs2_daddr_t fs_csaddr;   /* blk addr of cyl grp summary area */
        int64_t     fs_pendingblocks; /* blocks in process of being freed */
        int32_t     fs_pendinginodes; /* inodes in process of being freed */
        int32_t     fs_snapinum[FSMAXSNAP]; /* list of snapshot inode numbers */
        int32_t     fs_avgfilesize;   /* expected average file size */
        int32_t     fs_avgfpdir;      /* expected # of files per directory */
        int32_t     fs_save_cgsize;   /* save real cg size to use fs_bsize */
        int32_t     fs_sparecon32[26]; /* reserved for future constants */
        int32_t  fs_flags;        /* see FS_ flags below */
        int32_t     fs_contigsumsize; /* size of cluster summary array */
```

```
        int32_t     fs_maxsymlinklen; /* max length of an internal symlink */
        int32_t     fs_old_inodefmt;  /* format of on-disk inodes */
        uint64_t fs_maxfilesize;   /* maximum representable file size */
        int64_t     fs_qbmask;        /* ~fs_bmask for use with 64-bit size */
        int64_t     fs_qfmask;        /* ~fs_fmask for use with 64-bit size */
        int32_t     fs_state;         /* validate fs_clean field */
        int32_t     fs_old_postblformat; /* format of positional layout tables */
        int32_t     fs_old_nrpos;     /* number of rotational positions */
        int32_t     fs_spare5[2];     /* old fs_postbloff */
                            /* old fs_rotbloff */
        int32_t     fs_magic;         /* magic number */
};

/*
 * Filesystem identification
 */
#define    FS_UFS1_MAGIC 0x011954    /* UFS1 fast filesystem magic number */
#define    FS_UFS2_MAGIC 0x19540119  /* UFS2 fast filesystem magic number */
#define    FS_OKAY                   0x7c269d38  /* superblock checksum */
#define FS_42INODEFMT    -1     /* 4.2BSD inode format */
#define FS_44INODEFMT    2      /* 4.4BSD inode format */

/*
 * Preference for optimization.
 */
#define FS_OPTTIME          0           /* minimize allocation time */
#define FS_OPTSPACE         1           /* minimize disk fragmentation */
```

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of ‘blocks’. File system blocks of at most size MAXBSIZE can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be DEV_BSIZE, or some multiple of a DEV_BSIZE unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the **blksize**(*fs, ip, lbn*) macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 cannot be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it).

The *fs_minfree* element gives the minimum acceptable percentage of file system blocks that may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. The *fs_minfree* element may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of *fs_minfree* is 8%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 8, thus the default fragment size is an eighth of the block size.

The element *fs_optim* specifies whether the file system should try to minimize the time spent allocating blocks, or if it should attempt to minimize the space fragmentation on the disk. If the value of fs_minfree (see above) is less than 8%, then the file system defaults to optimizing for space to avoid running out of full sized blocks. If the value of minfree is greater than or equal to 8%, fragmentation is unlikely to be problematical, and the file system defaults to optimizing for time.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. With the default of 8 distinguished rotational positions, the resolution of the summary information is 2ms for a typical 3600 rpm drive.

The element *fs_old_rotdelay* gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for *fs_old_rotdelay* is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each NBPI bytes of disk space. The inode allocation strategy is extremely conservative.

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to (*struct cg*) must keep its size within MINBSIZE. Note that super-blocks are never more than size SBLOCKSIZE.

The path name on which the file system is mounted is maintained in *fs_fsmnt*. MAXMNTLEN defines the amount of space allocated in the super-block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. For a 4096 byte block size, it is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from *fs_csaddr* (size *fs_cssize*) in addition to the super-block.

N.B.: **sizeof**(*struct csum*) must be a power of two in order for the **fs_cs**() macro to work.

The *Super-block for a file system*: The size of the rotational layout tables is limited by the fact that the super-block is of size SBLOCKSIZE. The size of these tables is *inversely* proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (*fs_cpc*). The size of the rotational layout tables is derived from the number of bytes remaining in (*struct fs*).

The number of blocks of data per cylinder group is limited because cylinder groups are at most one block. The inode and free block tables must fit into a single block after deducting space for the cylinder group structure (*struct cg*).

The *Inode*: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is 'named' by its device/i-number pair. For further information, see the include file <*ufs/ufs/inode.h*>.

The format of an external attribute is defined by the extattr structure:

```
struct extattr {
        uint32_t ea_length;     /* length of this attribute */
        uint8_t  ea_namespace;          /* name space of this attribute */
        uint8_t  ea_contentpadlen;  /* bytes of padding at end of attribute */
        uint8_t  ea_namelength;         /* length of attribute name */
        char     ea_name[1];            /* attribute name (NOT nul-terminated) */
        /* padding, if any, to align attribute content to 8 byte boundary */
        /* extended attribute content follows */
```

```
};
```

Several macros are defined to manipulate these structures. Each macro takes a pointer to an extattr structure.

EXTATTR_NEXT(eap)	Returns a pointer to the next extended attribute following *eap*.
EXTATTR_CONTENT(eap)	Returns a pointer to the extended attribute content referenced by *eap*.
EXTATTR_CONTENT_SIZE(eap)	Returns the size of the extended attribute content referenced by *eap*.

The following code identifies an ACL:

```
    if (eap->ea_namespace == EXTATTR_NAMESPACE_SYSTEM &&
eap->ea_namelength == sizeof(POSIX1E_ACL_ACCESS_EXTATTR_NAME) - 1 &&
    strncmp(eap->ea_name, POSIX1E_ACL_ACCESS_EXTATTR_NAME,
sizeof(POSIX1E_ACL_ACCESS_EXTATTR_NAME) - 1) == 0) {
            aclp = EXTATTR_CONTENT(eap);
            acllen = EXTATTR_CONTENT_SIZE(eap);
            ...
    }
```

HISTORY

A super-block structure named filsys appeared in Version 6 AT&T UNIX. The file system described in this manual appeared in 4.2BSD.