NAME

fs_ufs, inode_ufs, inode – format of a ufs file system volume

SYNOPSIS

#include <sys/param.h>
#include <sys/types.h>
#include <sys/fs/ufs_fs.h>
#include <sys/fs/ufs_inode.h>

DESCRIPTION

Standard UFS file system storage volumes have a common format for certain vital information. Every volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 15 contain primary and secondary bootstrapping programs.

The actual file system begins at sector 16 with the super-block. The layout of the super-block is defined by the header **<sys/fs/ufs_fs.h>**.

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, and by the information in the cylinder group blocks. The super-block is critical data and is replicated before each cylinder group block 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.

fs_clean

fs_clean indicates the state of the file system. The **FSCLEAN** state indicates an undamaged, cleanly unmounted file system. The **FSACTIVE** state indicates a mounted file system that has been updated. The **FSSTABLE** state indicates an idle mounted file system. The **FSFIX** state indicates that this fs is mounted, contains inconsistent file system data and is being repaired by **fsck**. The **FSBAD** state indicates that this file system contains inconsistent file system data. It is not necessary to run **fsck** on any unmounted file systems with a state of **FSCLEAN** or **FSSTABLE**. **mount**(2) will return **ENOSPC** if a UFS file system with a state of **FSACTIVE** is being mounted for read-write.

To provide additional safeguard, **fs_clean** could be trusted only if **fs_state** contains a value equal to **FSOKAY** - **fs_time**, where **FSOKAY** is a constant integer. Otherwise, **fs_clean** is treated as though it contains the value of **FSACTIVE**.

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 exclusively of large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated 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; aligned fragments are examined to determine block availability.

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 *lost+found* directory is given the next available inode when it is initially created by **mkfs**(1M).

fs_minfree

fs_minfree gives the minimum acceptable percentage of file system blocks which may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. **fs_minfree** 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 10%.

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.

fs_optim

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** is less than 10%, then the file system defaults to optimizing for space to avoid running out of full sized blocks. If the value of **fs_minfree** is greater than or equal to 10%, 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. **fs_nrpos** is the number of rotational positions which are distinguished. With the default **fs_nrpos** of 8, the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay

fs_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_rotdelay** varies from drive to drive (see **tunefs**(1M)).

fs_maxcontig

fs_maxcontig gives the maximum number of blocks, belonging to one file, that will be allocated contiguously before inserting a rotational delay.

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 large enough to hold a cylinder group block, thus changes to **(struct cg)** must keep its size within **MINBSIZE**. Note: super-blocks are never more than size **SBSIZE**.

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**. 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.

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

The inode is the focus of all file activity in the 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 header **<sys/fs/ufs_inode.h>**.

SEE ALSO **fsck_ufs**(1M), **mkfs_ufs**(1M), **tunefs**(1M), **mount**(2)