NAME

fs – format of file system volume

DESCRIPTION

Caution: this information applies only to the latest versions of the UNIX system.

Every file system storage volume (e.g. RF disk, RK disk, RP disk, DECtape reel) has a common format for certain vital information. Every such volume is divided into a certain number of 256 word (512 byte) blocks. Block 0 is unused and is available to contain a bootstrap program, pack label, or other information.

Block 1 is the *super block.* Starting from its first word, the format of a super-block is

```
struct {
        int     isize;
        int     fsize;
        int     nfree;
        int     free[100];
        int     ninode;
        int     inode[100];
        char    flock;
        char    ilock;
        char    fmod;
        int     time[2];
};
```

Isize is the number of blocks devoted to the i-list, which starts just after the super-block, in block 2. *Fsize* is the first block not potentially available for allocation to a file. This number is unused by the system, but is used by programs like *check (1)* to test for bad block numbers. The free list for each volume is maintained as follows. The *free* array contains, in *free[1], ... , free[nfree−1],* up to 99 numbers of free blocks. *Free[0]* is the block number of the head of a chain of blocks constituting the free list. The first word in each free-chain block is the number (up to 100) of free-block numbers listed in the next 100 words of this chain member. The first of these 100 blocks is the link to the next member of the chain. To allocate a block: decrement *nfree,* and the new block is *free[nfree].* If the new block number is 0, there are no blocks left, so give an error. If *nfree* became 0, read in the block named by the new block number, replace *nfree* by its first word, and copy the block numbers in the next 100 words into the *free* array. To free a block, check if *nfree* is 100; if so, copy *nfree* and the *free* array into it, write it out, and set *nfree* to 0. In any event set *free[nfree]* to the freed block's number and increment *nfree.*

Ninode is the number of free i-numbers in the *inode* array. To allocate an i-node: if *ninode* is greater than 0, decrement it and return *inode[ninode].* If it was 0, read the i-list and place the numbers of all free inodes (up to 100) into the *inode* array, then try again. To free an i-node, provided *ninode* is less than 100, place its number into *inode[ninode]* and increment *ninode.* If *ninode* is already 100, don't bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

Flock and *ilock* are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of *fmod* on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

Time is the last time the super-block of the file system was changed, and is a double-precision representation of the number of seconds that have elapsed since 0000 Jan. 1 1970 (GMT). During a reboot, the *time* of the super-block for the root file system is used to set the system's idea of the time.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 32 bytes long, so 16 of them fit into a block. Therefore, i-node i is located in block $(i + 31) / 16$, and begins $32 \cdot ((i + 31) \pmod{16})$ bytes from its start. I-node 1 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node is as follows.

```
struct {
        int     flags;          /* +0: see below */
        char    nlinks;         /* +2: number of links to file */
        char    uid;            /* +3: user ID of owner */
```

```
        char    gid;            /* +4: group ID of owner */
        char    size0;          /* +5: high byte of 24-bit size */
        int     size1;          /* +6: low word of 24-bit size */
        int     addr[8];        /* +8: block numbers or device number */
        int     actime[2];      /* +24: time of last access */
        int     modtime[2];     /* +28: time of last modification */
};
```

The flags are as follows:

100000	i-node is allocated
060000	2-bit file type:
000000	plain file
040000	directory
020000	character-type special file
060000	block-type special file.
010000	large file
004000	set user-ID on execution
002000	set group-ID on execution
000400	read (owner)
000200	write (owner)
000100	execute (owner)
000070	read, write, execute (group)
000007	read, write, execute (others)

Special files are recognized by their flags and not by i-number. A block-type special file is basically one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files the high byte of the first address word specifies the type of device; the low byte specifies one of several devices of that type. The device type numbers of block and character special files overlap.

The address words of ordinary files and directories contain the numbers of the blocks in the file (if it is small) or the numbers of indirect blocks (if the file is large).

Byte number *n* of a file is accessed as follows. *N* is divided by 512 to find its logical block number (say *b*) in the file. If the file is small (flag 010000 is 0), then *b* must be less than 8, and the physical block number is *addr[b].*

If the file is large, *b* is divided by 256 to yield *i,* and *addr[i]* is the physical block number of the indirect block. The remainder from the division yields the word in the indirect block which contains the number of the block for the sought-for byte.

For block *b* in a file to exist, it is not necessary that all blocks less than *b* exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

SEE ALSO

check (VIII)