FILE SYSTEM(g) FILE SYSTEM(g)

NAME

fs – format of file system volume

DESCRIPTION

Caution: this information applies only to the latest versions of the MERT file 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 {
        char    *s_isize;       /* number of blocks of inodes */
        char    *s_fsize;       /* number of blocks in file system */
        struct  {
                char    *stblk;
                char    *ncblks;
        } s_ext[64];
        int     s_ninode;       /* number of free inodes */
        int     s_inode[60];
        int     s_nupdate;      /* number of update entries filled */
        struct  {
                char    *stblk;
                int     nublks;
        } s_update[30];
        char    s_flock;
        char    s_ilock;
        char    s_fmod;
        char    s_ronly;
        int     s_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 (d)* to test for bad block numbers.

The free list for each volume is maintained as follows. The in-core free list for each mounted volume consists of 64 double-word entries. The first word in an entry is the first free block *stblk* of the number *ncblks* of consecutive free blocks described by this extent. The free list for each volume is also maintained in a bit map kept on the volume starting just beyond the blocks devoted to the i-list. A number of update entries *s_nupdate* are kept in the in-core list *s_update* to keep the bit map uptodate. These update entries consist of double word entries, a starting block number and number of consecutive blocks.

To allocate *nb* blocks, the in-core free list is searched for the best entry to use. The algorithm used is to search the list for the smallest entry from which *nb* blocks can be allocated. If there are not *nb* free blocks, the largest entry is chosen. If there are no free blocks, the in-core free list is reconstructed using the bit map maintained on the volume. If there are still no free blocks, an error is returned. The in-core free list is updated and an entry is put in the update list. When the update list becomes full, the bit map is updated on the volume using the in-core update list and the update list is marked empty.

To free *nb* blocks, the in-core free list is searched according to the following algorithm. The *nb* blocks are added to an existing entry if they are contiguous with it. The entry is put in a null entry if one exists. If there is no empty entry, the smallest entry is replaced by the new entry. This entry is also put on the update list with a negative block count to distinguish it from an "alloc" entry. When the update list becomes full, the bit map is updated on the volume using the in-core update list and the update list is marked empty.

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. *Ronly* is a flag used to indicate that the file system is read-only, i.e. no files may be modified.

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 64 words long, so 4 of them fit into a block. Therefore, i-node *i* is located in block $(i + 7) / 4$, and begins $128*((i + 7) \pmod 4)$ 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    fill;                   /* +5: used internally */
        int     size0;                  /* +6: high byte of 32-bit size */
        int     size1;                  /* +8: low word of 32-bit size */
        struct {
                int *stblk;             /* starting block number */
                int *ncblks;            /* number of cons. blocks */
        } extents[27];
        int     actime[2];              /* +118: time of last access */
        int     modtime[2];             /* +122: time of last modification */
        int     chksum;                 /* not used */
};
```

The flags are as follows:

```
100000      i-node is allocated
070000      3-bit file type:
        000000      plain file
        040000      directory
        020000      character-type special file
```

060000 block-type special file
070000 record-type special file.
010000 contiguous 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 extent 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 extent words of ordinary files and directories contain pairs of starting block numbers and number of consecutive blocks. As many extents are used as are required to describe the discontinuous pieces of the file. A contiguous file requires only one extent.

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. The physical block number is the *b* th logical block in the list of extents. The remainder from the division yields the byte in the block which is to be accessed.

SEE ALSO

check (d)