

The BFS filesystem structure

The UnixWare Boot FileSystem (BFS) is a filesystem used in SCO UnixWare. It contains all files necessary for UnixWare boot procedures (such as `unix`). Because the object of the `bfs` filesystem type is to allow quick and simple booting, BFS was designed as a contiguous flat filesystem. It is not intended to support general users. The only directory `bfs` supports is the root directory. Users can create only regular files; no directories or special files can be created in the `bfs` filesystem.

A BFS filesystem consists of three parts:

- Superblock
- Inodes
- Data area

Each block on disk is 512 bytes long, blocks are numbered from zero. Most data structures use "offset from beginning of disk". Divide this number to get block number.

NOTE: Operations on a BFS filesystem in SCO UnixWare severely limited. For example, it is not possible to have two files open for writing simultaneously. These restrictions do not apply to operations involving only the reading of files.

You can read a BFS filesystem from your Linux box. See [BFS Linux module home page](#).

The BFS superblock

The superblock is at the beginning of disk, block 0.

Type	Name	Description
32bit int	magic	Magic number (0x1BADFACE)
32bit int	start	Start of data blocks (in bytes)
32bit int	size	Size of filesystem (in bytes)
4x 32bit int	sanity words	Sanity words are used to recover filesystem after interrupted compaction . They are usually 0xFFFFFFFF.

BFS inodes

The inode contains all the information about a file except its name. Filenames are kept in the root directory, the only directory in the BFS filesystem. An inode is 64 bytes long. Inode table starts at block

number 1 and fills the space between superblock and first data block (usually root directory). First inode has number 2.

Type	Name	Description
32bit int	inode number	Inode number, often contains "garbage" in high 16 bits.
32bit int	first block	First block of file. Next block is n+1, n+2, ... n+x.
32bit int	Last block	Last block of file
32bit int	offset to eof	Disk offset to end of file (in bytes)
32bit int	Attributes	File attributes (1 = regular file, 2 = directory)
32bit int	mode	File mode, rwxrwxrwx (only low 9 bits used)
32bit int	uid	File owner - user id
32bit int	gid	File owner - group id
32bit int	nlinks	Hard link count
32bit int	atime	Access time
32bit int	mtime	Modify time
32bit int	ctime	Create time
4x 32bit int	spare	Unused, should be zero

The number of inodes is defined when mkfs is used to create the filesystem.

BFS storage blocks

The remainder of the space allocated to the filesystem is taken up by data blocks. The storage blocks store the root directory and the regular files. For a regular file, the storage blocks contain the contents of the file. For the root directory, the storage blocks contain 16-byte entries.

Type	Name	Description
16bit int	inode	File inode number
14 8bit characters	name	File name

The root directory ***MUST*** begin with two entries "." and "..", both with inode number 2 (root directory).

Managing BFS data blocks

The data or storage blocks for a file are allocated contiguously. The data block after the last data block used in the filesystem is considered the next data block available to store a file. When a file is deleted, its data blocks are released.

Compaction

Compaction is a way of recovering data blocks by shifting files until the gaps left behind by deleted files are eliminated. This operation can be expensive, but it is necessary because of the method used by BFS to store and delete files. You need to perform compaction when either of the following situations occurs:

- The system has reached the end of the filesystem, and there are still free blocks available.
- The system deletes a large file and the file after it on disk is small and is the last file in the filesystem. (Small files are files of no more than ten blocks; large files are files of 500 or more blocks.)

Related links

[BFS Linux module](#)

[SCO homepage](#)

[Filesystems HOWTO](#)

Copyright (c) 1999 Martin Hinner, mhi@penguin.cz