# Optimized MPEG Filesystem (OMFS)

## Overview

OMFS is a filesystem created by SonicBlue for use in the ReplayTV DVR and Rio Karma MP3 player. The filesystem is extent-based, utilizing block sizes from 2k to 8k, with hash-based directories. This filesystem driver may be used to read and write disks from these devices.

Note, it is not recommended that this FS be used in place of a general filesystem for your own streaming media device. Native Linux filesystems will likely perform better.

More information is available at:

> http://linux-karma.sf.net/

Various utilities, including mkomfs and omfsck, are included with omfsprogs, available at:

> https://bobcopeland.com/karma/

Instructions are included in its README.

## Options

OMFS supports the following mount-time options:

| | |
|---|---|
| uid=n | make all files owned by specified user |
| gid=n | make all files owned by specified group |
| umask=xxx | set permission umask to xxx |
| fmask=xxx | set umask to xxx for files |
| dmask=xxx | set umask to xxx for directories |

## Disk format

OMFS discriminates between "sysblocks" and normal data blocks. The sysblock group consists of super block information, file metadata, directory structures, and extents. Each sysblock has a header containing CRCs of the entire sysblock, and may be mirrored in successive blocks on the disk. A sysblock may have a smaller size than a data block, but since they are both addressed by the same 64-bit block number, any remaining space in the smaller sysblock is unused.

Sysblock header information:

```
struct omfs_header {
        __be64 h_self;                  /* FS block where this is located */
        __be32 h_body_size;             /* size of useful data after header */
        __be16 h_crc;                   /* crc-ccitt of body_size bytes */
        char h_fill1[2];
        u8 h_version;                   /* version, always 1 */
        char h_type;                    /* OMFS_INODE_X */
        u8 h_magic;                     /* OMFS_IMAGIC */
        u8 h_check_xor;                 /* XOR of header bytes before this */
        __be32 h_fill2;
};
```

Files and directories are both represented by omfs_inode:

```
struct omfs_inode {
        struct omfs_header i_head;      /* header */
        __be64 i_parent;                /* parent containing this inode */
        __be64 i_sibling;               /* next inode in hash bucket */
        __be64 i_ctime;                 /* ctime, in milliseconds */
        char i_fill1[35];
        char i_type;                    /* OMFS_[DIR,FILE] */
        __be32 i_fill2;
        char i_fill3[64];
        char i_name[OMFS_NAMELEN];      /* filename */
        __be64 i_size;                  /* size of file, in bytes */
};
```

Directories in OMFS are implemented as a large hash table. Filenames are hashed then prepended into the bucket list beginning at OMFS_DIR_START. Lookup requires hashing the filename, then seeking across i_sibling pointers until a match is found on i_name. Empty buckets are represented by block pointers with all-1s (~0).

A file is an omfs_inode structure followed by an extent table beginning at OMFS_EXTENT_START:

```
struct omfs_extent_entry {
        __be64 e_cluster;               /* start location of a set of blocks */
        __be64 e_blocks;                /* number of blocks after e_cluster */
};

struct omfs_extent {
        __be64 e_next;                  /* next extent table location */
        __be32 e_extent_count;          /* total # extents in this table */
        __be32 e_fill;
        struct omfs_extent_entry e_entry;       /* start of extent entries */
};
```

Each extent holds the block offset followed by number of blocks allocated to the extent. The final extent in each table is a terminator with e_cluster being ~0 and e_blocks being ones'-complement of the total number of blocks in the table.

If this table overflows, a continuation inode is written and pointed to by e_next. These have a header but lack the rest of the inode structure.