Extended Introduction to the Minix File System

This is a description of the Minix v1 filesystem. It is based on the work by Scott Heavner, and extended by Arno Bakker.

> *This is a brief introduction to the Minix file system compiled by Scott D. Heavner (***sdh@po.cwru.edu***). Please direct any comments or corrections back to the author. You are free to distribute this document to anyone in any form as long as this notice remains intact.*

1 Definitions

block: Group of consecutive bytes on a disk. The size of the group is 1024 bytes (1 KiB) for the Minix v1 filesystem, denoted with the constant `BLOCK_SIZE` in this document. To the file system programmer, a disk looks like a sequence of blocks. The blocks on the disk are numbered starting at 0.

zone: mostly used interchangeably with block in this document. In the original FS design, a zone could consist of multiple blocks.

super block: the second block on a disk which contains information about the type and size of the file system.

inode: Stores all the information about a file or directory except its name.

2 Physical Layout

Minix v1 FS Layout		
	Size (Blocks)	Description
Boot Block	1	Reserved for OS boot code.
Super Block	1	Info about the file system.
Inode Map	#Inodes/8/BLOCK_SIZE	Keeps track of used/unused inodes.
Zone Map	#Data Zones/8/BLOCK_SIZE	Keeps track of used/unused zones.
Inode Table	(32 × #Inodes)/BLOCK_SIZE	Store info about files/devices.
Data Zones	Many	File/Directory contents.

3 Super Block

The superblock for the Minix v1 FS is offset 1024 bytes from the start of the disk, this is to leave room for boot code, like LILO or GRUB. The superblock basically details the size of the file system. It contains the number of inodes, number of data zones, space used by the inode and zone map, the zone size of the filesystem and a two byte magic number indicating the version of the Minix file system.

It is documented in <linux/minix_fs.h> (Isn't source code wonderful?):

```
struct minix_super_block {
        unsigned short s_ninodes;       /* Number of inodes */
```

```
        unsigned short s_nzones;        /* Number of data zones */
        unsigned short s_imap_blocks;   /* Space used by inode map (blocks) */
        unsigned short s_zmap_blocks;   /* Space used by zone map  (blocks) */
        unsigned short s_firstdatazone; /* First zone with ``file'' data */
        unsigned short s_log_zone_size; /* Size of a data zone =
                                               (1024 << s_log_zone_size) */
        unsigned long s_max_size;       /* Maximum file size (bytes) */
        unsigned short s_magic;         /* Minix 14/30 ID number */
        unsigned short s_state;         /* Mount state, was it
                                                    cleanly unmounted */
};
```

If we translate this to a layout on disk, it looks as follows:

Minix v1 super block																
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x00	No. inodes		No. zones		Imap blocks		ZMap blocks		First data zone		Log zone size		Max. file size			
0x10	Minix magic		Mount state													

The "Minix magic" number can have the following values in Minix v1:

MINIX_SUPER_MAGIC	0x137F	original Minix v1 FS, names are 14 chararacters long
MINIX_SUPER_MAGIC2	0x138F	Minix v1 FS, names are 30 chararacters long

4 Inodes

The inode contains all the important information about a file or directory, except its name. It contains the file permissions, file type, user, group, size, modification time, number of links, and the location and order of all the blocks in the file, in the zone array. On disk, all values are stored in low byte – high byte order.

From <linux/minix_fs.h>

```
struct minix_inode {
        unsigned short i_mode;
        unsigned short i_uid;
        unsigned long i_size;
        unsigned long i_time;
        unsigned char i_gid;
        unsigned char i_nlinks;
        unsigned short i_zone[9];
};
```

Minix Inode Entry																
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x00	MODE		UID		SIZE				TIME				GID	LINKS	ZONE 0	
0x10	ZONE 1		ZONE 2		ZONE 3		ZONE 4		ZONE 5		ZONE 6		ZONE 7		ZONE 8	

Now, for the zone array. The first seven zone numbers (0-6) are the numbers of the blocks on the disk that directly contain the file (or directory) data. The eighth is the block number of an **indirect block**; this block does not contain file data, but 512 (`BLOCK_SIZE/2`) numbers of the blocks that do. The ninth zone number in the inode points to a **double indirect block**. The double indirect block contains 512 pointers to more indirect blocks, each of which points to 512 data zones. For small files, only the 7 direct numbers in the inode will be used to hold the list of blocks that store the contents of the file, for larger files the indirect and double indirect block have to be used.

Each indirect block adds 1 KiB to the amount of data needed to store the file. It's no big deal, but it's nice that small files (under 8k) don't have this overhead. Technically, you can make 262M files ($7 + 512 + 512 \times 512$ k, see also `s_max_size` in the superblock information), but since the Minix v1 FS uses unsigned shorts for block numbers, it is limited to 64M partitions.

To determine the meaning of the `mode` field, consult <linux/types.h>. Below is a list of octal numbers which can be extracted from the mode entry. This information can also be found in the stat(2) and chmod(2) man pages.

```
#define S_IFSOCK 0140000 /* Socket */
#define S_IFLNK  0120000 /* Symbolic Linux */
#define S_IFREG  0100000 /* Regular file */
#define S_IFBLK  0060000 /* Block device */
#define S_IFDIR  0040000 /* Directory */
#define S_IFCHR  0020000 /* Character device */
#define S_IFIFO  0010000 /* FIFO/Named pipe */
#define S_ISUID  0004000 /* Set user id upon execution */
#define S_ISGID  0002000 /* Set group id upon execution */
#define S_ISVTX  0001000 /* Sticky bit */

#define S_IRUSR 00400 /* User readable */
#define S_IWUSR 00200 /* User writable */
#define S_IXUSR 00100 /* User executable */

#define S_IRGRP 00040 /* Group readable */
#define S_IWGRP 00020 /* Group writable */
#define S_IXGRP 00010 /* Group executable */

#define S_IROTH 00004 /* World/Other readable */
#define S_IWOTH 00002 /* World/Other writable */
#define S_IXOTH 00001 /* World/Other executable */
```

5 Directories

An inode can refer either to a file or a directory. In case of a directory, the blocks in the zone array contain a list of directory entries in the following format.

From <linux/minix_fs.h>

```
struct minix_dir_entry {
```

```
        unsigned short inode;
        char name[0];
};
```

Minix Directory Entry			
	00	01	02 – 1F
0x00	INODE NUMBER		char[14 or 30] FILENAME

The directory entry associates a filename with an inode. The first two bytes of a directory entry indicate the inode, the remainder is the filename. On the Minix v1 FS, the root directory has inode number 1 (`MINIX_ROOT_INO`) (and not 0, although it is the first inode on the disk (also see below)). Depending on the filename size used, you can fit either 64 or 32 directory entries in a 1 KiB block which can exist anywhere in the data zone.

Hard links are made when two entries point to the same node. Soft links are stored in the data zone, indexed by an inode entry. A hard links uses only a directory entry, whereas a soft link uses a directory entry, an inode, and at least one data zone.

6 Inode and Zone Map

The filesystem must keep track of which inodes and zones are used. This is needed to find free space to put the information about a new file (inode) and its contents (zones). The Minix v1 filesystem uses bitmaps to keep track of which inodes and zones are used. These bitmaps are stored in the "Inode Map" and "Zone Map" blocks that sit between the super block and the "Inode Table" on the disk.

The "Inode Map" bitmap contains 1 bit for every inode, including inode 0. For the zone map it is more complicated, see below. In both cases, the bitmap is stored on disk as a sequence of bytes. For large bitmaps this takes up multiple blocks. Note bits are numbered from right-to-left in a byte. So a byte with bit 0 enabled looks like this (and has value 0x01) on disk:

Bit number	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	1

6.1 Zone Map Peculiarities

Quoting `http://oldlinux.org/Linux.old/study/linux-travel/MINIX-1.5/1.5/Source/commands/mkfs.c.gz`:

> "Unfortunately, the coding [of] the bit maps is inconsistent. The rules are:
>
> - For inodes: Every i-node occupies a bit map slot, even i-node 0. The first i-node [actually stored in the "Inode Table"] on the disk is i-node 1, not 0.
> - For zones: Zone map bit 0 is for the last i-node block on disk. The first zone available goes with bit 1 in the map.
>
> Thus for i-nodes, every i-node, starting at 0 occupies a bit map slot, but for zones, only those starting with the final i-node block occupy bit slots. This is inconsistent. In

> retrospect it would might have been simpler to have bit 0 of the zone map be zone 0 on the disk. Although this would have increased the zone bit map by a few dozen bits, it would have prevented a number of bugs in the early days. This is an example of what happens when one ignores the maxim: First make it work, then make it optimal. For both maps, 0 = available, 1 = in use."

In other words, the "Inode Map" starts with a bit for inode 0, but this inode is not stored in the "Inode Table" on disk. The first entry in the "Inode Table" is for inode 1.

Bit 0 in the "Zone Map" denotes the zone with number *s_firstdatazone* minus 1. The field *s_firstdatazone* is a field in the superblock that denotes the first zone used to store actual file and directory data. As a consequence, bit 1 in the "Zone Map" denotes the *s_firstdatazone* block itself.