

Operating System

Dr. GuoJun LIU

Harbin Institute of Technology

<http://os.guojunhit.cn>

硬盘分区表

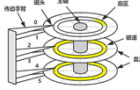
表 9-10 硬盘主引导扇区 MBR 的结构

偏移位置	名称	长度 (字节)	说明
0x000	MBR 代码	446	引导程序代码和数据。
0x1BE	分区表项 1	16	第 1 个分区表项, 共 16 字节。
0x1CE	分区表项 2	16	第 2 个分区表项, 共 16 字节。
0x1DE	分区表项 3	16	第 3 个分区表项, 共 16 字节。
0x1EE	分区表项 4	16	第 4 个分区表项, 共 16 字节。
0x1FE	引导标志	2	有效引导扇区的标志, 值分别是 0x55, 0xAA。

主引导扇区

表 9-11 硬盘分区表项结构

位置	名称	大小	说明
0x00	boot_ind	字节	说明标志, 4 个分区中只能有一个分区是可引导的。 0x00=从该分区引导操作系统; 0x01=从该分区引导操作系统。
0x01	head	字节	分区起始处的头字节。头字节范围为 0-255。
0x02	sector	字节	分区起始的当前柱面区号 (柱 0-65) 和柱面高 2 高 (柱 0-67)。
0x03	cyl	字节	分区起始的柱面区号低 8 位。柱面号范围为 0-1023。
0x04	sys_ind	字节	分区类型字节。0x00-D0S, 0x01-D0S-Old, 0x02-386-Linux...
0x05	end_head	字节	分区结束处的头字节。头字节范围为 0-255。
0x06	end_sector	字节	分区结束的当前柱面区号 (柱 0-65) 和柱面高 2 高 (柱 0-67)。
0x07	end_cyl	字节	分区结束的柱面区号低 8 位。柱面号范围为 0-1023。
0x08-0x0fb	start_sector	长字节	分区起始的物理扇区号。它对整个硬盘的扇区号顺序从 0 开始计数。
0x0c-0x0fb	nr_sects	长字节	分区占用的扇区数。



Dr. GuoJun LIU

Operating System

Slides-4

Chapter A4

Minix

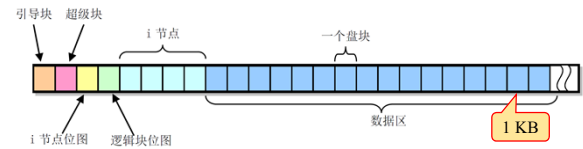
Minix 1.0

Minix

主引导扇区



硬盘设备上的分区和文件系统



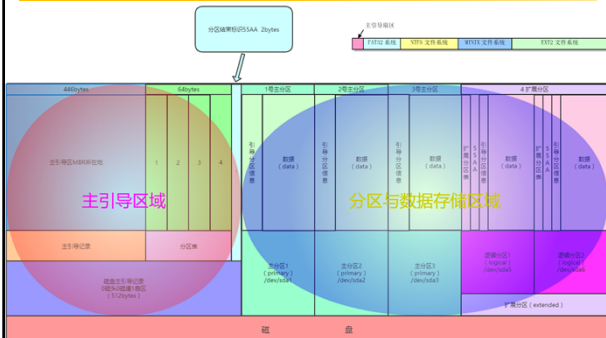
建有 MINIX 文件系统的块设备上各部分的布局示意图

Dr. GuoJun LIU

Operating System

Slides-5

磁盘存储结构逻辑图

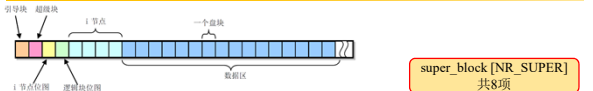


Dr. GuoJun LIU

Operating System

Slides-3

MINIX 文件系统的超级块结构



super_block[NR_SUPER]
共8项

```

124: struct super_block {
125:     unsigned short s_ninodes;
126:     unsigned short s_size;
127:     unsigned short s_inap_blocks;
128:     unsigned short s_nmap_blocks;
129:     unsigned short s_firstdatazone;
130:     unsigned short s_log_zone_size;
131:     unsigned long s_max_size;
132:     unsigned short s_magic;
133:     /* These are only in memory */
134:     struct buffer_head *s_inap;
135:     struct buffer_head *s_nmap;
136:     unsigned short s_dev;
137:     struct inode *s_isup;
138:     struct inode *s_imount;
139:     unsigned long s_time;
140:     struct task_struct *s_wait;
141:     unsigned char s_lock;
142:     unsigned char s_err;
143:     unsigned char s_orly;
144: } = end super_block =

```

变量名	数据类型	说明
<code>n_idnodes</code>	short	1 号节点数
<code>n_mnodes</code>	short	逻辑块数 (或称为块区数)
<code>n_imp_blocks</code>	short	1 号节点图所占块数
<code>n_imp_blocks</code>	short	逻辑块在图所占块数
<code>n_findname</code>	short	数据区 (即一个逻辑块) 的
<code>n_log_maxsize</code>	short	Log (值域) 数据长度
<code>n_max_size</code>	long	最大文件长度
<code>n_magic</code>	short	文件系统标识 (0x1f37)
<code>n_imp[8]</code>	buffer_head *	1 号节点图在高速缓冲指针数组
<code>n_imp[8]</code>	buffer_head *	逻辑块位置在高速缓冲指针数组
<code>n_dev</code>	short	设备块号
<code>n_dev</code>	n_inode *	设备文件在设备上的 1 号节点
<code>n_inount</code>	n_inode *	设备文件系统安装到的 1 号节点
<code>n_time</code>	long	块访问时间
<code>n_wait</code>	task_struct *	等待本超级块的设备进程
<code>n_lock</code>	char	锁标志
<code>n_rd_only</code>	char	只读标志
<code>n_wrt_only</code>	char	只写标志 (写) 标志

Dr. GuoJun LIU

Operating System

Slides-6

```

242: void mount_root(void)
243: {
244:     int i, free;
245:     struct super_block * p;
246:     struct m_inode * mi;
247:
248:     if (32 != sizeof(struct d_inode))
249:         panic("bad i-node size");
250:     for(i=0; i<NR_FILE; i++)
251:         file_table[i].f_count=0;
252:     if (MAJOR(ROOT_DEV) == 2) {
253:         printk("Insert root floppy and press ENTER");
254:         wait_for_keypress();
255:     }
256:     for(p = &super_block[0]; p < &super_block[NR_SUPER]; p++) {
257:         p->s_dev = 0;
258:         p->s_lock = 0;
259:         p->s_wait = NULL;
260:     }
261:     if (!p->read_super(ROOT_DEV))
262:         panic("unable to read root inode");
263:     if (!p->iget(ROOT_DEV, ROOT_INO))
264:         panic("unable to read root inode");
265:     mj->i_count += 3; /* NOTE! it is logically used a times, not 1 */
266:     p->s_isup = p->s_inount = mj;
267:     current->pid = mj;
268:     current->root = mj;
269:     free=0;
270:     lep->s_nzones;
271:     while (--i >= 0)
272:         if (!set_bit(188191, p->s_zmap[i]>>13)->b_data))
273:             free++;
274:     print("%d/%d free blocks\n", free, p->s_nzones);
275:     free=0;
276:     lep->s_ninodes++;
277:     while (--i >= 0)
278:         if (!set_bit(188191, p->s_imap[i]>>13)->b_data))
279:             free++;
280:     print("%d/%d free inodes\n", free, p->s_ninodes);
281: } = end mount_root =

```

```

100: struct super_block {
101:     unsigned short s_nzones;
102:     unsigned short s_inap_blocks;
103:     unsigned short s_inap_blocks;
104:     unsigned short s_firstdatazone;
105:     unsigned long s_log_zone_size;
106:     unsigned char s_magic;
107:     /* these are only for memory use */
108:     struct buffer_head * s_inmap;
109:     struct super_block * s_inmap;
110:     unsigned short s_dev;
111:     struct task_struct * s_wait;
112:     struct task_struct * s_wait;
113:     unsigned char s_lock;
114:     unsigned char s_dir;
115:     /* end super_block */
116: }
117:
118: struct m_inode {
119:     unsigned short i_mode;
120:     unsigned short i_uid;
121:     unsigned long i_size;
122:     unsigned long i_atime;
123:     unsigned long i_ctime;
124:     unsigned char i_gid;
125:     unsigned char i_nlinks;
126:     /* these are in memory also */
127:     struct task_struct * i_wait;
128:     unsigned long i_atime;
129:     unsigned long i_ctime;
130:     unsigned short i_dev;
131:     unsigned short i_count;
132:     unsigned short i_lock;
133:     unsigned char i_dir;
134:     unsigned char i_pipe;
135:     unsigned char i_mount;
136:     unsigned char i_seek;
137:     unsigned char i_update;
138: }
139:
140: #define NAME_LEN 14
141: #define ROOT_INO 1
142: #define I_MAP_SLOTS 8
143: #define SUPER_MAGIC 0x137f
144: #include <linux/fs.h>

```

kernel/blk_drv/hd.c

逻辑块位图

i节点位图

8191

include/linux/fs.h

Slides-7

Minix 1.0 的 i 节点结构

字段名称	数据类型	说明
i_mode	short	文件的类型和属性 (rev 位)
i_uid	short	文件所有者的用户 ID
i_size	long	文件长度 (字节)
i_atime	long	修改时间 (从 1970.1.1.0 时算起, 秒)
i_gid	char	文件组主的组 ID
i_nlinks	char	链接数 (多少个文件目录项指向该 i 节点)
i_zone[9]	short	文件所占用的盘上逻辑块号数组, 其中: zone[0]-zone[6] 是直接块号; zone[7] 是一次间接块号; zone[8] 是二次 (双重) 间接块号。 注: zone 是区的意思, 可译成区或逻辑块。 对于设备特殊文件名的 i 节点, 其 zone[0] 中存放的是该设备名所指向的设备号。
i_wait	task_struct *	等待该 i 节点的进程。
i_atime	long	最后访问时间。
i_ctime	long	i 节点自身被修改时间。
i_dev	short	i 节点所在的设备号。
i_count	short	i 节点被引用的次数, 0 表示空闲。
i_lock	char	i 节点被锁定标志。
i_dir	char	i 节点已被修改 (脏) 标志。
i_pipe	char	i 节点用作管道标志。
i_mount	char	i 节点安装了其他文件系统标志。
i_seek	char	搜索标志 (iseek 操作时)。
i_update	char	i 节点已更新标志。

在盘上和内存中的字节, 共 32 字节

又在内存中使用的字节

直接块

一次间接块

二次间接块

三次间接块

Dr. GuoJun LIU

Operating System

Slides-10

逻辑块位图

- 每个位依次代表盘上数据区中的一个逻辑块
- 位0闲置不用, 初始为1
- 位1代表盘上数据区中第一个数据盘块

```

100: struct super_block {
101:     unsigned short s_nzones;
102:     unsigned short s_inap_blocks;
103:     unsigned short s_inap_blocks;
104:     unsigned short s_firstdatazone;
105:     unsigned long s_log_zone_size;
106:     unsigned char s_magic;
107:     /* these are only for memory use */
108:     struct buffer_head * s_inmap;
109:     struct super_block * s_inmap;
110:     unsigned short s_dev;
111:     struct task_struct * s_wait;
112:     struct task_struct * s_wait;
113:     unsigned char s_lock;
114:     unsigned char s_dir;
115:     /* end super_block */
116: }
117:
118: struct m_inode {
119:     unsigned short i_mode;
120:     unsigned short i_uid;
121:     unsigned long i_size;
122:     unsigned long i_atime;
123:     unsigned long i_ctime;
124:     unsigned char i_gid;
125:     unsigned char i_nlinks;
126:     /* these are in memory also */
127:     struct task_struct * i_wait;
128:     unsigned long i_atime;
129:     unsigned long i_ctime;
130:     unsigned short i_dev;
131:     unsigned short i_count;
132:     unsigned short i_lock;
133:     unsigned char i_dir;
134:     unsigned char i_pipe;
135:     unsigned char i_mount;
136:     unsigned char i_seek;
137:     unsigned char i_update;
138: }
139:
140: #define NAME_LEN 14
141: #define ROOT_INO 1
142: #define I_MAP_SLOTS 8
143: #define SUPER_MAGIC 0x137f
144: #include <linux/fs.h>

```

引导块

超级块

i 节点

一个盘块

数据区

1024

8块 * 1k 字节/块 * 8位/字节 = 64k 位

64k * 1k = 64M

最多使用8块缓冲块 s_zmap[8]

Dr. GuoJun LIU

Operating System

Slides-8

Minix 1.0 的 i 节点结构

字段名称	数据类型	说明
i_mode	short	文件的类型和属性 (rev 位)
i_uid	short	文件所有者的用户 ID
i_size	long	文件长度 (字节)
i_atime	long	修改时间 (从 1970.1.1.0 时算起, 秒)
i_gid	char	文件组主的组 ID
i_nlinks	char	链接数 (多少个文件目录项指向该 i 节点)
i_zone[9]	short	文件所占用的盘上逻辑块号数组, 其中: zone[0]-zone[6] 是直接块号; zone[7] 是一次间接块号; zone[8] 是二次 (双重) 间接块号。 注: zone 是区的意思, 可译成区或逻辑块。 对于设备特殊文件名的 i 节点, 其 zone[0] 中存放的是该设备名所指向的设备号。
i_wait	task_struct *	等待该 i 节点的进程。
i_atime	long	最后访问时间。
i_ctime	long	i 节点自身被修改时间。
i_dev	short	i 节点所在的设备号。
i_count	short	i 节点被引用的次数, 0 表示空闲。
i_lock	char	i 节点被锁定标志。
i_dir	char	i 节点已被修改 (脏) 标志。
i_pipe	char	i 节点用作管道标志。
i_mount	char	i 节点安装了其他文件系统标志。
i_seek	char	搜索标志 (iseek 操作时)。
i_update	char	i 节点已更新标志。

在盘上和内存中的字节, 共 32 字节

又在内存中使用的字节

直接块

一次间接块

二次间接块

三次间接块

Dr. GuoJun LIU

Operating System

Slides-11

i 节点位图和 i 节点

- 每个位代表一个 i 节点
- i 节点位0闲置不用, 初始为1
- i 节点部分存放着文件或目录的索引节点
- 每个文件或目录都有一个 i 节点
- i 节点结构存放着对应文件的相关信息, 32 字节

Dr. GuoJun LIU

Operating System

Slides-9

通过文件名找对应文件磁盘块位置

文件目录

i 节点部分

其他字段

一次间接块

二次间接块

三次间接块

四次间接块

五次间接块

六次间接块

七次间接块

八次间接块

九次间接块

十次间接块

十一次间接块

十二次间接块

十三次间接块

十四次间接块

十五次间接块

十六次间接块

十七次间接块

十八次间接块

十九次间接块

二十次间接块

二十一次间接块

二十二次间接块

二十三次间接块

二十四次间接块

二十五次间接块

二十六次间接块

二十七次间接块

二十八次间接块

二十九次间接块

三十次间接块

三十一次间接块

三十二次间接块

三十三次间接块

三十四次间接块

三十五次间接块

三十六次间接块

三十七次间接块

三十八次间接块

三十九次间接块

四十次间接块

四一次间接块

四二次间接块

四三次间接块

四四次间接块

四五次间接块

四六次间接块

四七次间接块

四八次间接块

四九次间接块

五一次间接块

五二次间接块

五三次间接块

五四次间接块

五五次间接块

五六次间接块

五七次间接块

五八次间接块

五九次间接块

六一次间接块

六二次间接块

六三次间接块

六四次间接块

六五次间接块

六六次间接块

六七次间接块

六八次间接块

六九次间接块

七一次间接块

七二次间接块

七三次间接块

七四次间接块

七五次间接块

七六次间接块

七七次间接块

七八次间接块

七九次间接块

八一次间接块

八二次间接块

八三次间接块

八四次间接块

八五次间接块

八六次间接块

八七次间接块

八八次间接块

八九次间接块

九一次间接块

九二次间接块

九三次间接块

九四次间接块

九五次间接块

九六次间接块

九七次间接块

九八次间接块

九九次间接块

一百次间接块

Dr. GuoJun LIU

Operating System

Slides-12

i 节点表 数据块 目录块



Operating System

Slides-13

Dr. GuoJun LIU

Operating System

Slides-16



Operating System

Slides-14



Operating System

Slides-17

```

152:     return NULL;
153: } // and find_entry is

```

100

Slides

Slides-18

The screenshot shows a terminal window with the following commands and output:

```

guojun@ubuntu: ~/myfs
File Edit View Search Terminal Help
guojun@ubuntu:~/myfs$ dd if=/dev/zero of=minix.lng bs=1k count=360
660+0 records out
668640 bytes (369 kB, 360 KiB) copied, 0.008803917 s, 459 MB/s
guojun@ubuntu:~/myfs$ mkfs.minix minix.lng
128 inodes
360 blocks
tstdev=tzzone=8 (8)
onesize=1024
taxsize=268966912
guojun@ubuntu:~/myfs$

```

A red callout box contains the command:

```
sudo mount minix.img -o loop mnt
```

The terminal window also shows a detailed disk layout for the minix.lng file, including file names, sizes, and block counts.

对比数据区块位图

引导块 超级块

i节点

d节点

数据区

i节点位图 逻辑块位图

0x43
0100 0011
1和6在使用

数据区有44*8=352 盘块

```

*
00000c00 43 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | C.....|
00000c10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000c20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 fe ff ff ff 00 00 00 00 00 00 00 00 | .....|
00000c30 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....|
*

```

Dr. GuoJun LIU

Operating System

Slides-21

对比超级块

```

*
00000400  00 00 68 01 01 00 01 00 08 00 00 00 00 1c 08 10 |..h.....|
00000410  8f 13 01 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000420  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*

struct d_super_block {
    unsigned short   s_ninodes;           //0x0080, 十进制128, inode总共128个, 4K
    unsigned short   s_nzones;           //0x0168, 十进制360, 总共360个zone
    unsigned short   s_imap_blocks;      //0x0001, 十进制1, inode位图占1个块
    unsigned short   s_zmap_blocks;      //0x0001, 十进制1, zone位图占1个块
    unsigned short   s_firstdatazone;    //0x0008, 十进制8, 第一个数据区编号是8
    unsigned short   s_log_zone_size;    //0x0000, log表示的一块数据大小, 1kb
    unsigned long    s_max_size;         //0x10081c00, 十进制268966912, 最大文件大小
    unsigned short   s_magic;            //0x138f, minix魔数
};

```

Dr. GuoJun LIU

Operating System

Slides-20

对比i节点数据

```

*
00001000 ed 41 e8 03 80 00 00 00 46 99 e7 5e e8 02 08 00 |.A.....F..^...|
00001010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00001020 00 00 e8 03 00 10 00 00 1e 99 e7 5e e8 00 09 00 |.....^...|
00001030 0a 00 0b 00 0c 00 00 00 00 00 00 00 00 00 00 00 |.....|
00001040 a4 81 e8 03 04 00 00 00 46 99 e7 5e e8 01 0d 00 |.....F..^...|
00001050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*

```

0x1000 ~ 0x2000 4k空间为i节点空间

```

struct minode {
    unsigned short i_mode;    // 0x41ed, 040755, 目录文件, rwxr-xr-x
    unsigned short i_uid;    // 0x03e8, 1000
    unsigned long i_size;     // 0x00000080, 128
    unsigned long i_mtime;    // 0x5ee79946
    unsigned char i_gid;      // 0xe8
    unsigned char i_nlinks;    // 0x02
    unsigned short i_zone[9]; // 0x08, i_zone[0]=8, 数据块在第8号区块
};

```

Dr. GuoJun LIU

Operating System

Slides-23

对比i节点位图

The diagram illustrates the relationship between i-nodes, logical blocks, and physical sectors. It shows a horizontal bar representing disk space, divided into three main sections: a small header area (orange), a series of colored blocks (pink, yellow, green, cyan) representing logical blocks, and a large blue section representing physical sectors. Labels indicate that the first four colored blocks are 'i nodes' and the rest are 'one disk block'. Below the bar, arrows point from specific bits in the i-node bitmap to their corresponding logical blocks and then to the physical sectors. A red box highlights that the 0x0b bit (decimal 11) corresponds to the 16th logical block (16 * 8 = 128 sectors). A second red box notes that the 1st and 3rd bits are currently in use.

引导块 超级块

i 节点 一个盘块

i 节点位图 逻辑块位图 数据区

0x0b
0000 1011
1和3在使用

i 节点位图对应 $16 \times 8 = 128$ 个 i 节点

*
00000800 0b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ||
00000810 fe ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ||
00000820 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ||
*

Dr. GuoJun LIU Operating System Slides-21

[illegible]

Demo 2

[illegible]

命令汇总

- `dd if=/dev/zero of=minix.img bs=1k count=360`
- `mkfs.minix minix.img`
- `sudo mount minix.img -o loop mnt`
- `sudo umount mnt`
- `hexdump -C minix.img > hex-minix.txt`
- `stat aaa.txt` 查看某个文件inode信息
- `df -i | grep myfs` 查找带myfs的硬盘分区inode信息
- `ls -li` 查看对应的inode号