Operating System

Dr. GuoJun LIU

Harbin Institute of Technology

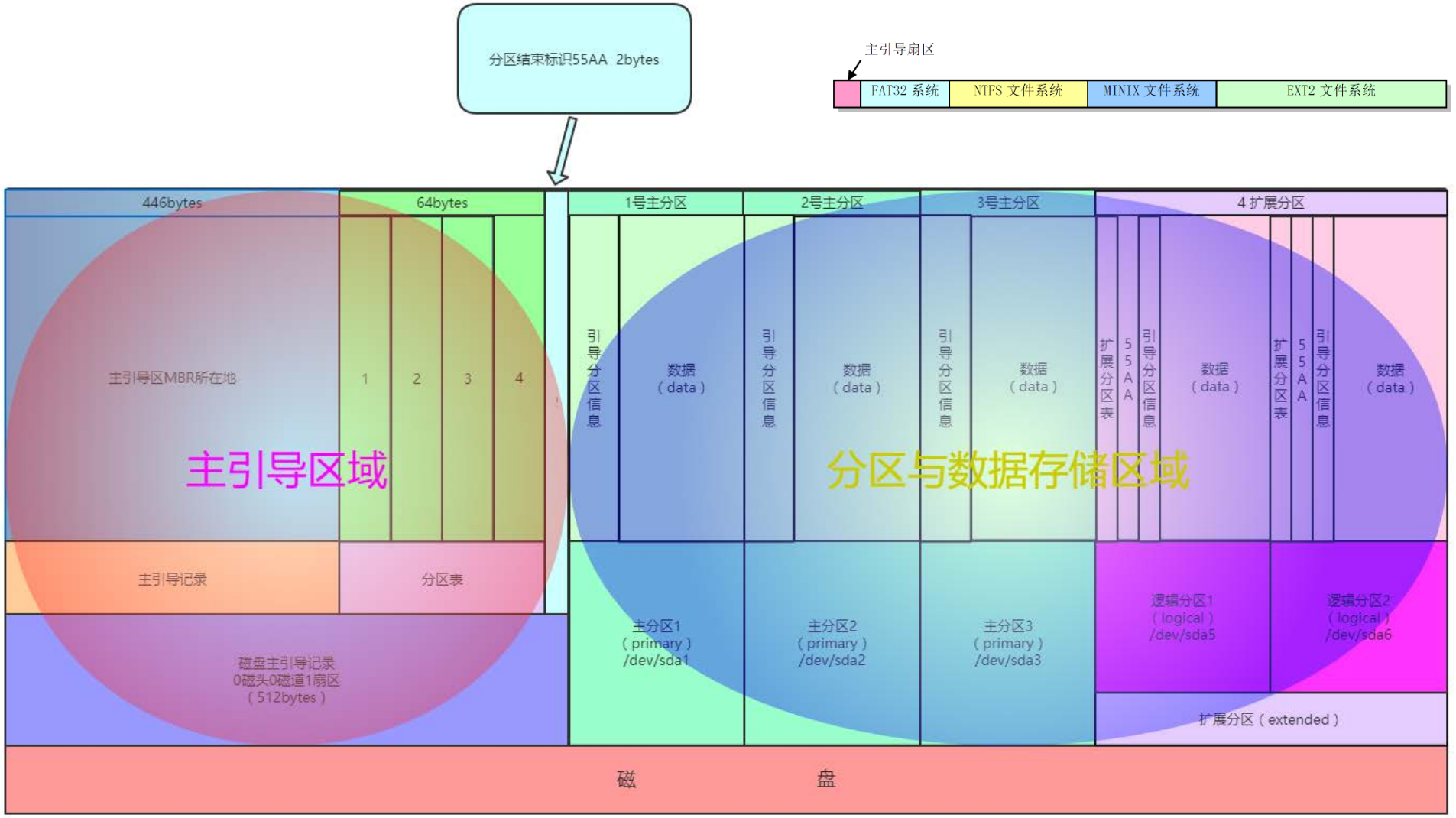
http://os.guojunhit.cn

Chapter A4

Minix

Minix 1.0

磁盘存储结构逻辑图


分区结束标识55AA 2bytes
主引导扇区
FAT32 系统
NTFS 文件系统
MINIX 文件系统
EXT2 文件系统
446bytes
64bytes
1号主分区
2号主分区
3号主分区
4 扩展分区
主引导区MBR所在地
1
2
3
4
引导分区信息
数据 (data)
扩展分区表
55AA
主引导区域
分区与数据存储区域
主引导记录
分区表
磁盘主引导记录
0磁头0磁道1扇区
(512bytes)
主分区1
(primary)
/dev/sda1
主分区2
(primary)
/dev/sda2
主分区3
(primary)
/dev/sda3
逻辑分区1
(logical)
/dev/sda5
逻辑分区2
(logical)
/dev/sda6
扩展分区 (extended)
磁 盘

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

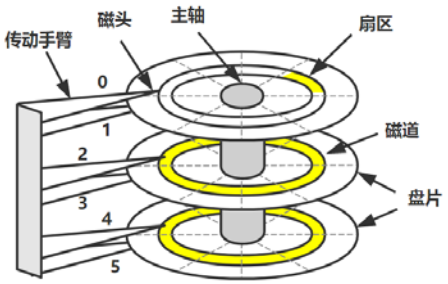
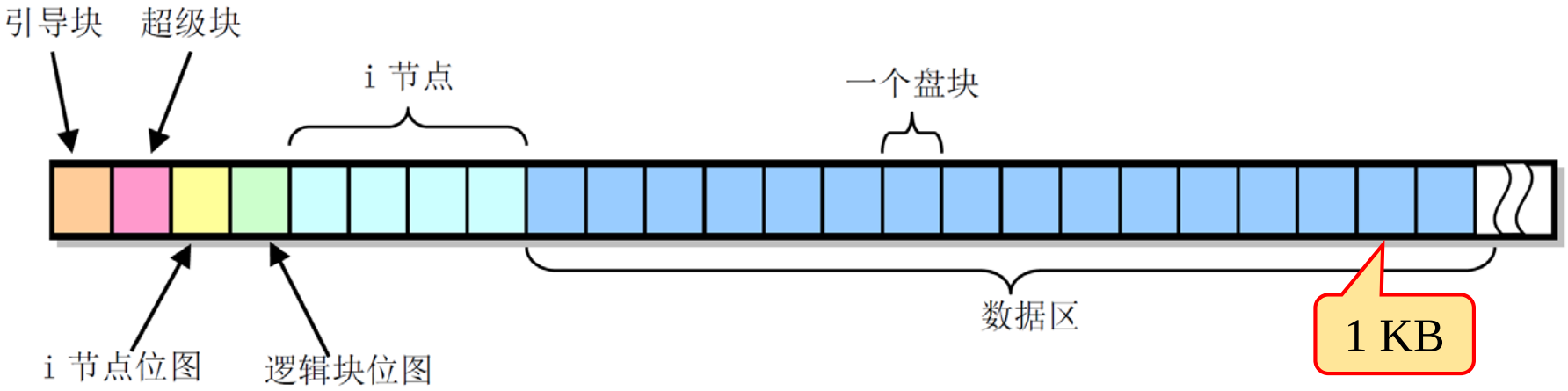


表 9－11 硬盘分区表项结构

位置	名称	大小	说明
0x00	boot_ind	字节	引导标志。4 个分区中同时只能有一个分区是可引导的。 0x00-不从该分区引导操作系统；0x80-从该分区引导操作系统。
0x01	head	字节	分区起始处的磁头号。磁头号范围为 0--255。
0x02	sector	字节	分区起始的当前柱面中扇区号(位 0-5) 和柱面号高 2 位(位 6-7)。
0x03	cyl	字节	分区起始处的柱面号低 8 位。柱面号范围为 0--1023。
0x04	sys_ind	字节	分区类型字节。0x0b-DOS; 0x80-Old Minix; 0x83-Linux …
0x05	end_head	字节	分区结束处的磁头号。磁头号范围为 0--255。
0x06	end_sector	字节	分区结束的当前柱面中扇区号(位 0-5) 和柱面号高 2 位(位 6-7)。
0x07	end_cyl	字节	分区结束的柱面号低 8 位。柱面号范围为 0--1023。
0x08-0x0b	start_sect	长字	分区起始的物理扇区号。它以整个硬盘的扇区号顺序从 0 计起。
0x0c-0x0f	nr_sects	长字	分区占用的扇区数。

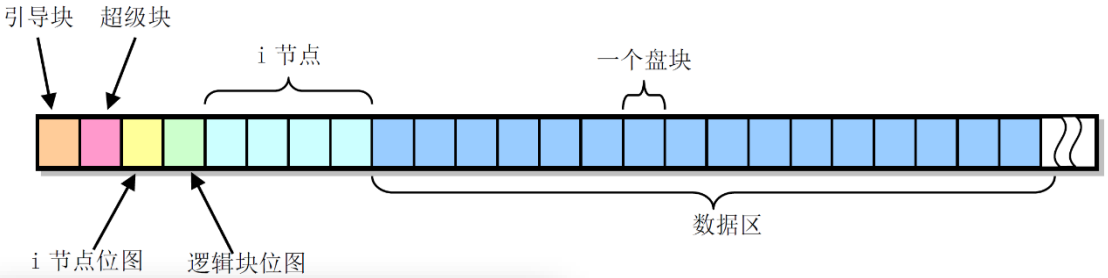
Minix

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

MINIX 文件系统的超级块结构




```
124: struct super_block {
125:     unsigned short s_ninodes;
126:     unsigned short s_nzones;
127:     unsigned short s_imap_blocks;
128:     unsigned short s_zmap_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_imap[8];
135:     struct buffer_head * s_zmap[8];
136:     unsigned short s_dev;
137:     struct m_inode * s_isup;
138:     struct m_inode * s_imount;
139:     unsigned long s_time;
140:     struct task_struct * s_wait;
141:     unsigned char s_lock;
142:     unsigned char s_rd_only;
143:     unsigned char s_dirt;
144: } « end super_block » ;
```

super.c的mount_root()中初始化

super_block [NR_SUPER]
共8项

	字段名称	数据类型	说明
出现在盘上和内存中的字段	s_ninodes	short	i 节点数
	s_nzones	short	逻辑块数(或称为区块数)
	s_imap_blocks	short	i 节点位图所占块数
	s_zmap_blocks	short	逻辑块位图所占块数
	s_firstdatazone	short	数据区中第一个逻辑块块号
	s_log_zone_size	short	Log_2(磁盘块数/逻辑块)
	s_max_size	long	最大文件长度
	s_magic	short	文件系统幻数 (0x137f)
仅在内存中使用的字段	s_imap[8]	buffer_head *	i 节点位图在高速缓冲块指针数组
	s_zmap[8]	buffer_head *	逻辑块位图在高速缓冲块指针数组
	s_dev	short	超级块所在设备号
	s_isup	m_inode *	被安装文件系统的根目录 i 节点
	s_imount	m_inode *	该文件系统被安装到的 i 节点
	s_time	long	修改时间
	s_wait	task_struct *	等待本超级块的进程指针
	s_lock	char	锁定标志
	s_rd_only	char	只读标志
	s_dirt	char	已被修改(脏)标志

kerne/blk_drv/hd.c

```
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 mount root");
263:     if (!(mi=iget(ROOT_DEV,ROOT_INO)))
264:         panic("Unable to read root i-node");
265:     mi->i_count += 3 ;  /* NOTE! it is logically used 4 times, not 1
266:     p->s_isup = p->s_imount = mi;
267:     current->pwd = mi;
268:     current->root = mi;
269:     free=0;
270:     i=p->s_nzones;
271:     while (-- i >= 0)
272:         if (!set_bit(i&8191,p->s_zmap[i>>13]->b_data))
273:             free++;
274:     printk("%d/%d free blocks\n\r",free,p->s_nzones);
275:     free=0;
276:     i=p->s_ninodes+1;
277:     while (-- i >= 0)
278:         if (!set_bit(i&8191,p->s_imap[i>>13]->b_data))
279:             free++;
280:     printk("%d/%d free inodes\n\r",free,p->s_ninodes);
281: } « end mount_root »
```

逻辑块位图

i节点位图

8191

```
124: struct super_block {
125:     unsigned short s_ninodes;
126:     unsigned short s_nzones;
127:     unsigned short s_imap_blocks;
128:     unsigned short s_zmap_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_imap[8];
135:     struct buffer_head * s_zmap[8];
136:     unsigned short s_dev;
137:     struct m_inode * s_isup;
138:     struct m_inode * s_imount;
139:     unsigned long s_time;
140:     struct task_struct * s_wait;
141:     unsigned char s_lock;
142:     unsigned char s_rd_only;
143:     unsigned char s_dirt;
144: } « end super_block » ;
```

```
93: struct m_inode {
94:     unsigned short i_mode;
95:     unsigned short i_uid;
96:     unsigned long i_size;
97:     unsigned long i_mtime;
98:     unsigned char i_gid;
99:     unsigned char i_nlinks;
100:    unsigned short i_zone[9];
101: /* these are in memory also */
102:    struct task_struct * i_wait;
103:    unsigned long i_atime;
```

```
68: struct buffer_head {
69:     char * b_data;          /* pointer to data block (1024 bytes) */
70:     unsigned long b_blocknr;    /* block number */
71:     unsigned short b_dev;       /* device (0 = free) */
72:     unsigned char b_uptodate;
73:     unsigned char b_dirt;       /* 0-clean,1-dirty */
74:     unsigned char b_count;      /* users using this block */
75:     unsigned char b_lock;       /* 0 - ok, 1 -locked */
76:     struct task_struct * b_wait;
77:     struct buffer_head * b_prev;
78:     struct buffer_head * b_next;
79:     struct buffer_head * b_prev_free;
80:     struct buffer_head * b_next_free;
81: };
```

```
36: #define NAME_LEN 14
37: #define ROOT_INO 1
38:
39: #define I_MAP_SLOTS 8
40: #define Z_MAP_SLOTS 8
41: #define SUPER_MAGIC 0x137F
```

include/linux/fs.h

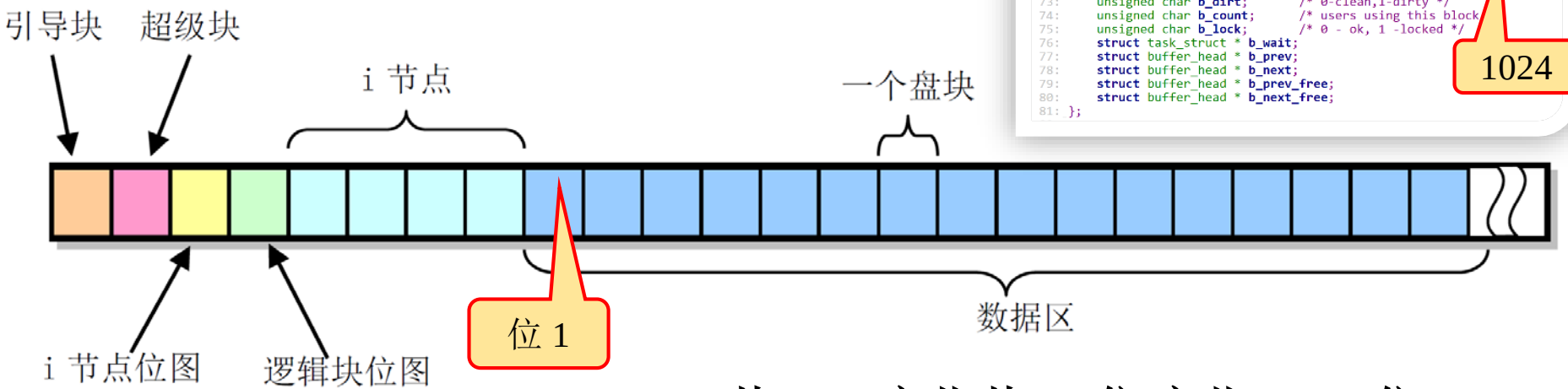
逻辑块位图

- 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

```
struct super_block {
    unsigned short s_ninodes;
    unsigned short s_nzones;
    unsigned short s_imap_blocks;
    unsigned short s_zmap_blocks;
    unsigned short s_firstdatazone;
    unsigned short s_log_zone_size;
    unsigned long s_max_size;
    unsigned short s_magic;
/* These are only in memory */
    struct buffer_head * s_imap[8];
    struct buffer_head * s_zmap[8];
    unsigned short s_dev;
    struct m_inode * s_isup;
    struct m_inode * s_imount;
    unsigned long s_time;
    struct task_struct * s_wait;
    unsigned char s_lock;
```

```
struct buffer_head {
    char * b_data;          /* pointer to data block (1024 bytes) */
    unsigned long b_blocknr;    /* block number */
    unsigned short b_dev;       /* device (0 = free) */
    unsigned char b_uptodate;
    unsigned char b_dirt;       /* 0-clean,1-dirty */
    unsigned char b_count;      /* users using this block */
    unsigned char b_lock;       /* 0 - ok, 1 -locked */
    struct task_struct * b_wait;
    struct buffer_head * b_prev;
    struct buffer_head * b_next;
    struct buffer_head * b_prev_free;
    struct buffer_head * b_next_free;
};
```

1024



最多使用8块缓冲块 s_zmap[8]

8块 * 1k 字节/块 * 8位/字节 = 64k 位

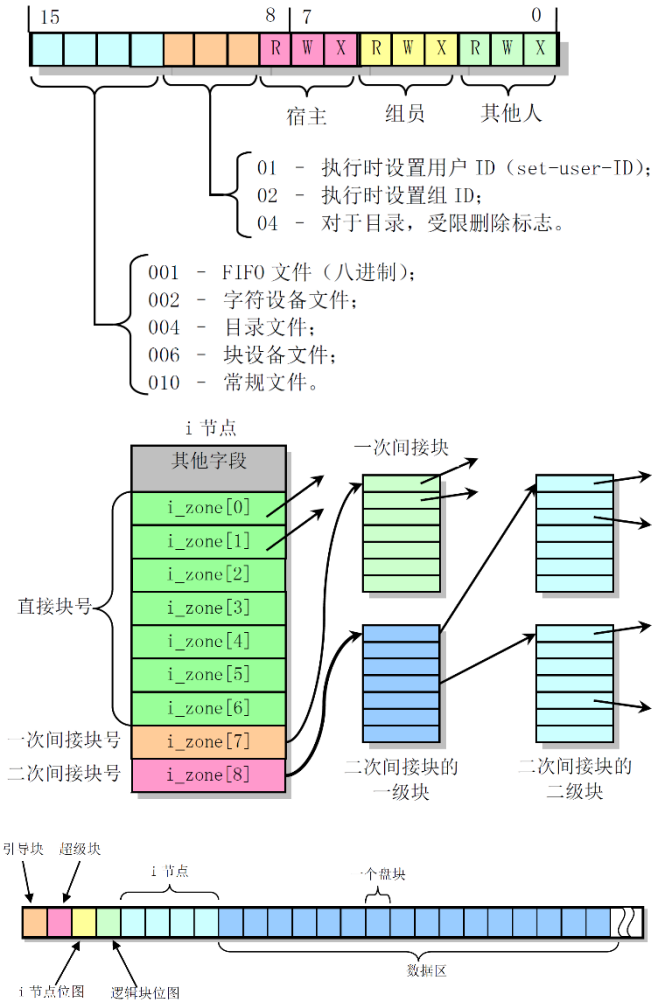
64k * 1k = 64M

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

Minix 1.0 的 i 节点结构

	字段名称	数据类型	说明
在盘上和内存中的字段，共 32 字节	i_mode	short	文件的类型和属性（rwx 位）
	i_uid	short	文件宿主的用户 id
	i_size	long	文件长度（字节）
	i_mtime	long	修改时间（从 1970.1.1:0 时算起，秒）
	i_gid	char	文件宿主的组 id
	i_nlinks	char	链接数（有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，可译成区块或逻辑块。 对于设备特殊文件名的 i 节点，其 zone[0]中存放的是该文件名所指设备的设备号。
仅在内存中使用的字段	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_num	short	i 节点号。
	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_dirt	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（lseek 操作时）。
	i_update	char	i 节点已更新标志。



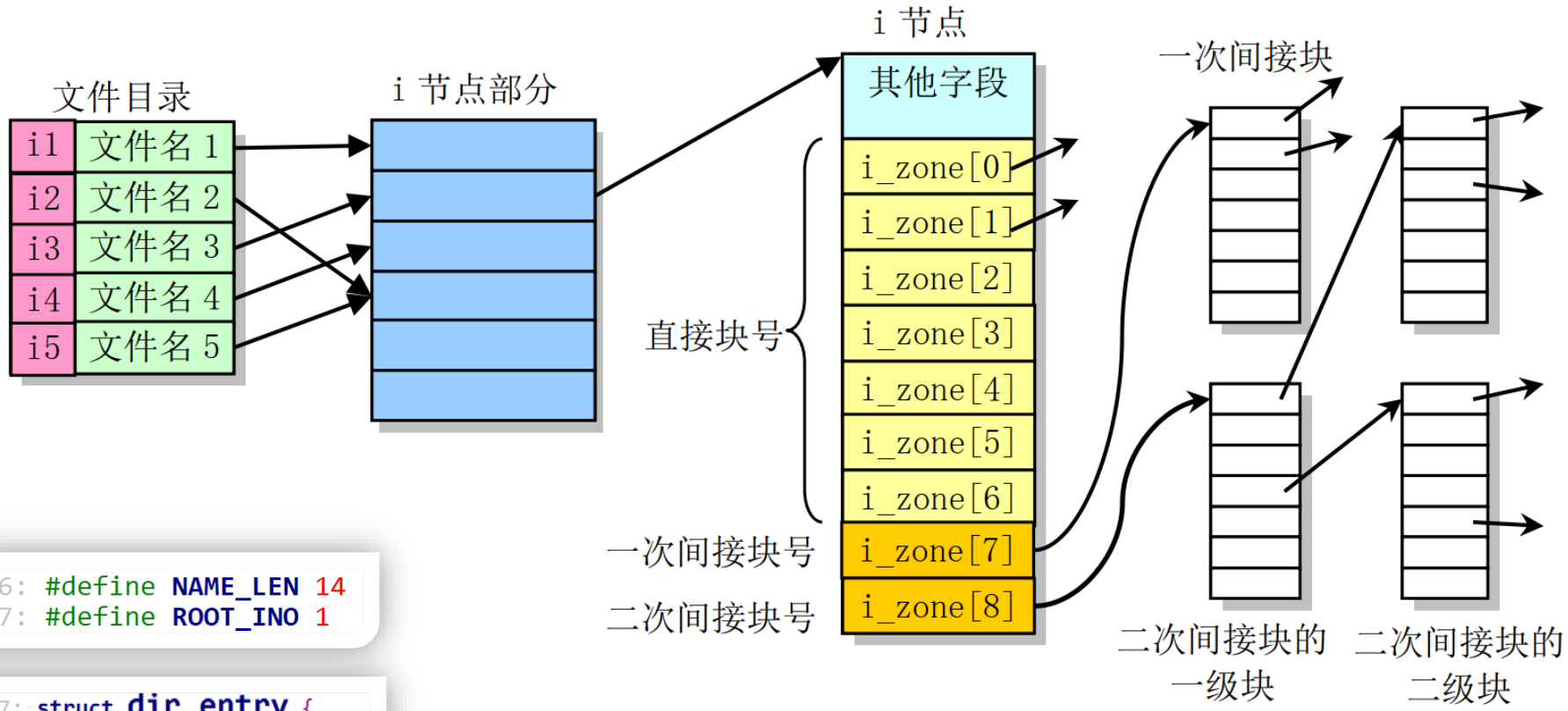
Minix 1.0 的 i 节点结构

	字段名称	数据类型	说明
在盘上和内存中的字段，共 32 字节	i_mode	short	文件的类型和属性（rwx 位）
	i_uid	short	文件宿主的用户 id
	i_size	long	文件长度（字节）
	i_mtime	long	修改时间（从 1970.1.1:0 时算起，秒）
	i_gid	char	文件宿主的组 id
	i_nlinks	char	链接数（有多少个文件目录项指向该 i 节
	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，可译成区块或逻辑 对于设备特殊文件名的 i 节点，其 zone[存放的是该文件名所指设备的设备号。
仅在内存中使用的字段	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_num	short	i 节点号。
	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_dirt	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（lseek 操作时）。
	i_update	char	i 节点已更新标志。

include\linux\fs.h

```
 93: struct m_inode {
 94:     unsigned short i_mode;
 95:     unsigned short i_uid;
 96:     unsigned long i_size;
 97:     unsigned long i_mtime;
 98:     unsigned char i_gid;
 99:     unsigned char i_nlinks;
100:     unsigned short i_zone[9];
101: /* these are in memory also */
102:     struct task_struct * i_wait;
103:     unsigned long i_atime;
104:     unsigned long i_ctime;
105:     unsigned short i_dev;
106:     unsigned short i_num;
107:     unsigned short i_count;
108:     unsigned char i_lock;
109:     unsigned char i_dirt;
110:     unsigned char i_pipe;
111:     unsigned char i_mount;
112:     unsigned char i_seek;
113:     unsigned char i_update;
114: } « end m_inode » ;
```

通过文件名找对应文件磁盘块位置

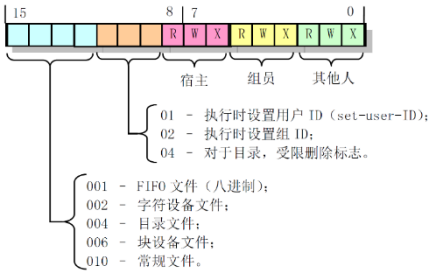


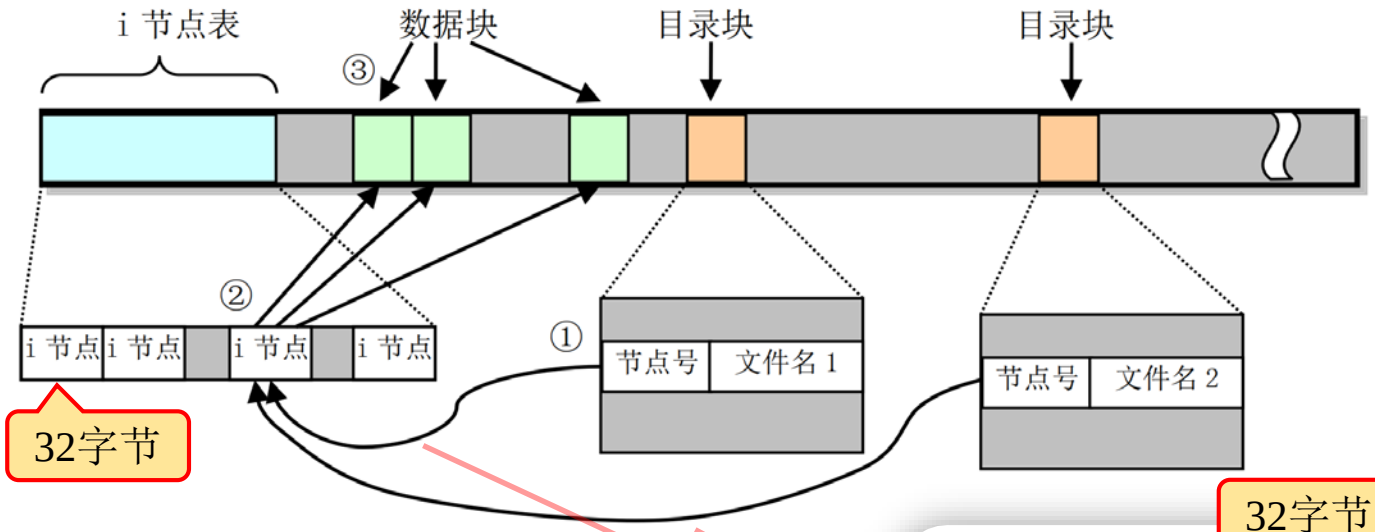

```
36: #define NAME_LEN 14
37: #define ROOT_INO 1
```

```
157: struct dir_entry {
158:     unsigned short inode;
159:     char name[NAME_LEN];
160: };
```

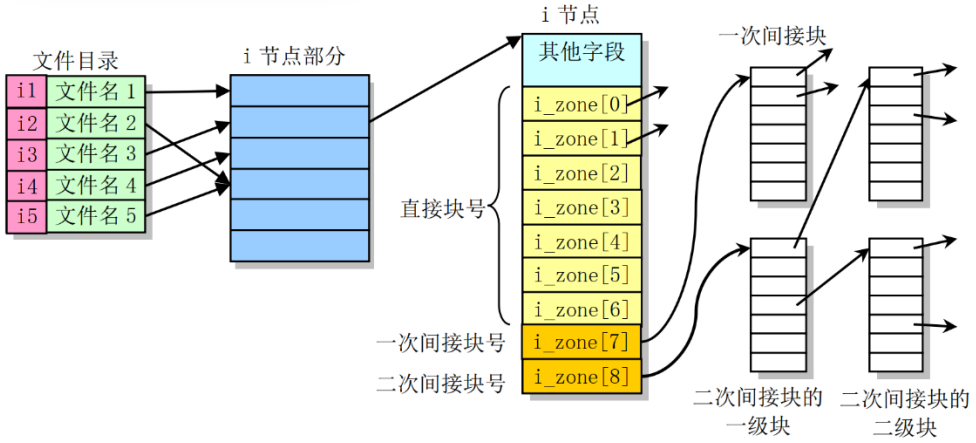
16字节

从文件名获取其数据块






```
157: struct dir_entry {
158:     unsigned short inode;
159:     char name[NAME_LEN];
160: };
```

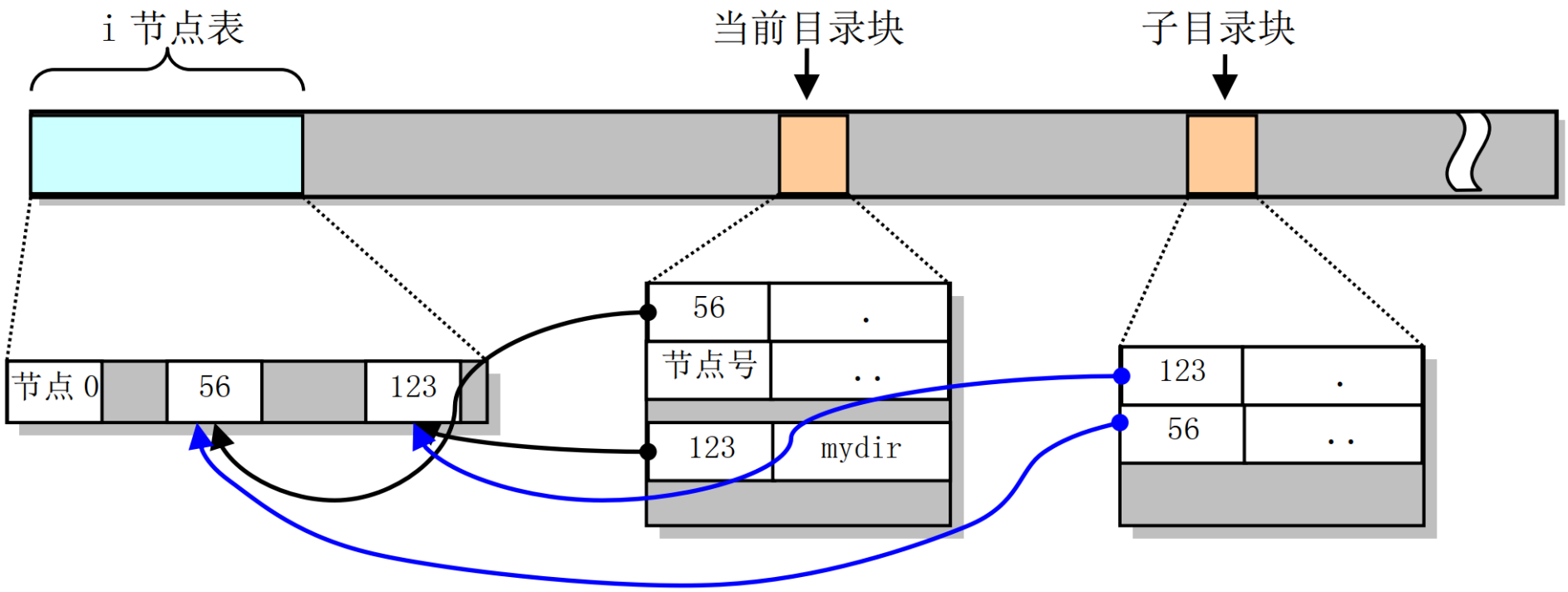


i_nlinks = 2

32字节

```
93: struct m_inode {
94:     unsigned short i_mode;
95:     unsigned short i_uid;
96:     unsigned long i_size;
97:     unsigned long i_mtime;
98:     unsigned char i_gid;
99:     unsigned char i_nlinks;
100:    unsigned short i_zone[9];
101: /* these are in memory also */
102:    struct task_struct * i_wait;
103:    unsigned long i_atime;
104:    unsigned long i_ctime;
105:    unsigned short i_dev;
106:    unsigned short i_num;
107:    unsigned short i_count;
108:    unsigned char i_lock;
109:    unsigned char i_dirt;
110:    unsigned char i_pipe;
111:    unsigned char i_mount;
112:    unsigned char i_seek;
113:    unsigned char i_update;
114: } « end m_inode » ;
```

目录文件目录项和子目录链接



include\linux\fs.h

```c
static struct buffer_head * find_entry(struct m_inode ** dir,
    const char * name, int namelen, struct dir_entry ** res_dir)
{
    int entries;
    int block,i;
    struct buffer_head * bh;
    struct dir_entry * de;
    struct super_block * sb;

#ifdef NO_TRUNCATE
    if (namelen > NAME_LEN)
        return NULL;
#else
    if (namelen > NAME_LEN)
        namelen = NAME_LEN;
#endif
    entries = (*dir)->i_size / (sizeof (struct dir_entry));
    *res_dir = NULL;
    if (!namelen)
        return NULL;
/* check for '..', as we might have to do some "magic" for it */
    if (namelen==2 && get_fs_byte(name)=='.' && get_fs_byte(name+1)=='.') {
/* '..' in a pseudo-root results in a faked '.' (just change namelen) */
        if ((*dir) == current->root)
            namelen=1;
        else if ((*dir)->i_num == ROOT_INO) {
/* '..' over a mount-point results in 'dir' being exchanged for the mounted
   directory-inode. NOTE! We set mounted, so that we can iput the new dir */
            sb=get_super((*dir)->i_dev);
            if (sb->s_imount) {
                iput(*dir);
                (*dir)=sb->s_imount;
                (*dir)->i_count++;
            }
        }
    }
    if (!(block = (*dir)->i_zone[0]))
        return NULL;
    if (!(bh = bread((*dir)->i_dev,block)))
        return NULL;
    i = 0;
    de = (struct dir_entry *) bh->b_data;
    while (i < entries) {
        if ((char *)de >= BLOCK_SIZE+bh->b_data) {
            brelse(bh);
            bh = NULL;
            if (!(block = bmap(*dir,i/DIR_ENTRIES_PER_BLOCK)) ||
                !(bh = bread((*dir)->i_dev,block))) {
                i += DIR_ENTRIES_PER_BLOCK;
                continue;
            }
            de = (struct dir_entry *) bh->b_data;
        }
        if (match(namelen,name,de)) {
            *res_dir = de;
            return bh;
        }
        de++;
        i++;
    }
    brelse(bh);
    return NULL;
} « end find_entry »
```

文件长度

目录项数

遍历

fs\nameic

```c
struct m_inode {
    unsigned short i_mode;
    unsigned short i_uid;
    unsigned long i_size;
    unsigned long i_mtime;
    unsigned char i_gid;
    unsigned char i_nlinks;
    unsigned short i_zone[9];
/* these are in memory also */
    struct task_struct * i_wait;
    unsigned long i_atime;
    unsigned long i_ctime;
    unsigned short i_dev;
    unsigned short i_num;
    unsigned short i_count;
    unsigned char i_lock;
    unsigned char i_dirt;
    unsigned char i_pipe;
    unsigned char i_mount;
    unsigned char i_seek;
    unsigned char i_update;
} « end m_inode » ;
```

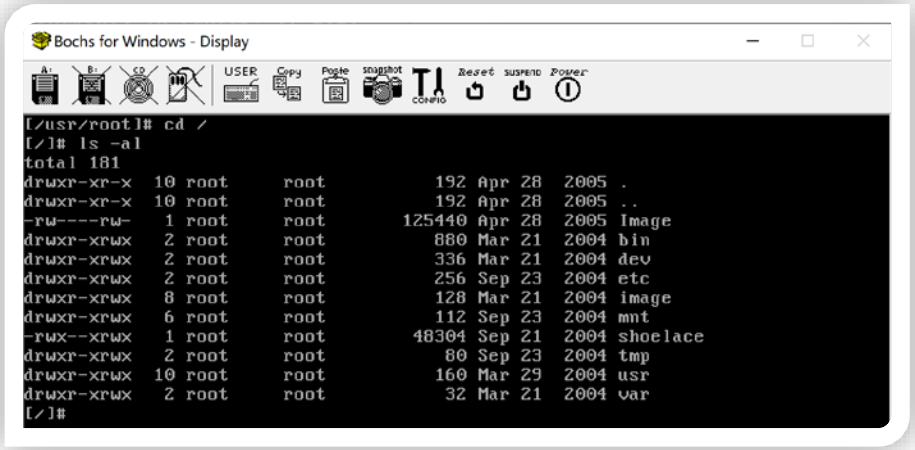
16字节

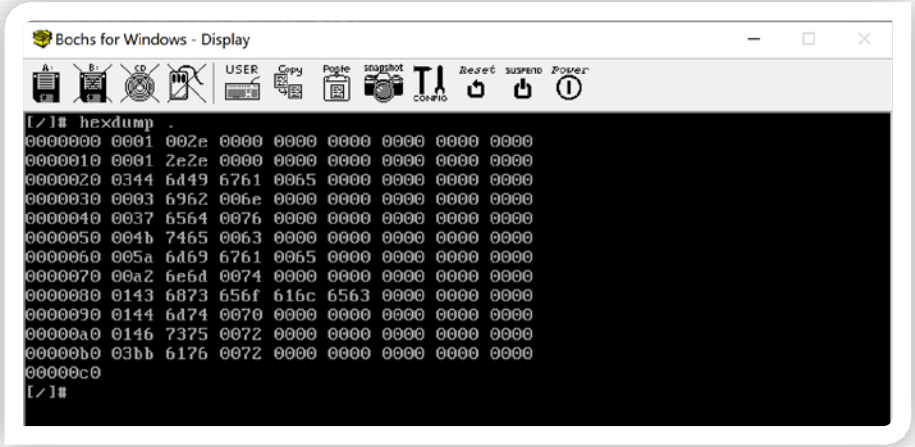
```c
struct dir_entry {
    unsigned short inode;
    char name[NAME_LEN];
};
```

```c
static int match(int len,const char * name,struct dir_entry * de)
{
    register int same ;

    if (!de || !de->inode || len > NAME_LEN)
        return 0;
    if (len < NAME_LEN && de->name[len])
        return 0;
    __asm__("cld\n\t"
        "fs ; repe ; cmpsb\n\t"
        "setz %%al"
        :"=a" (same)
        :"0" (0),"S" ((long) name),"D" ((long) de->name),"c" (len)
        );
    return same;
}
```

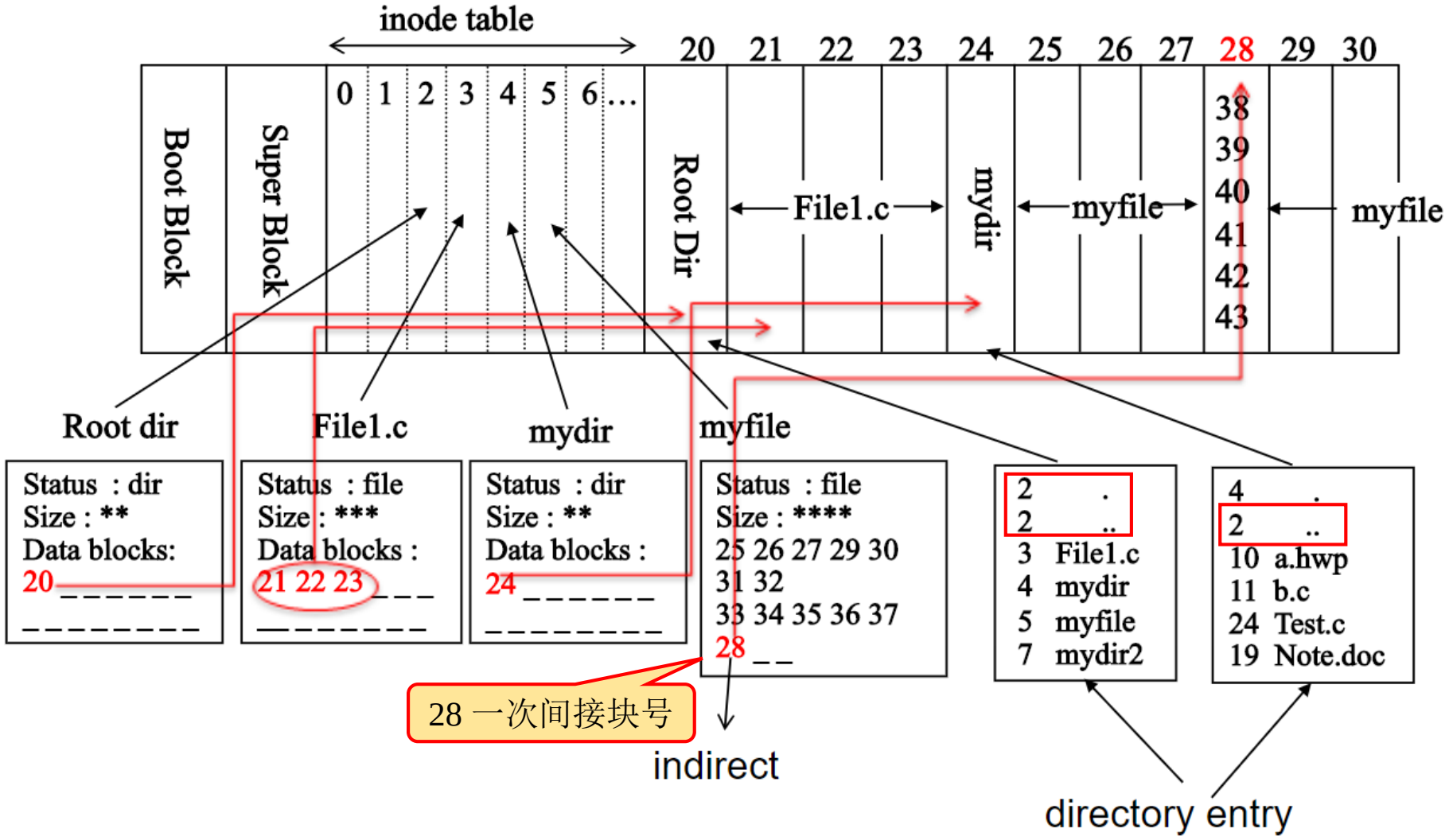
hexdump 查看目录内容

```
[/]# ls etc -la
total 32
drwxr-xr-x   2 root     root          224 Mar 21  2004 .
drwxr-xr-x  10 root     root          176 Mar 21  2004 ..
-rw-r--r--   1 root     root          137 Mar  4  2004 group
-rw-r--r--   1 root     root        11801 Mar  4  2004 magic
-rw-r--r--   1 root     root           11 Jan 22 18:12 mtab
-rw-r--r--   1 root     root          142 Mar  5  2004 mtools
-rw-r--r--   1 root     root          266 Mar  4  2004 passwd
-rw-r--r--   1 root     root          147 Mar  4  2004 profile
-rw-r--r--   1 root     root           57 Mar  4  2004 rc
-rw-r--r--   1 root     root         1034 Mar  4  2004 termcap
-rwx--x--x   1 root     root        10137 Jan 15  1992 update

[/]# hexdump etc
0000000 0004 002e 0000 0000 0000 0000 0000 0000        // .
0000010 0001 2e2e 0000 0000 0000 0000 0000 0000        // ..
0000020 0007 6372 0000 0000 0000 0000 0000 0000        // rc
0000030 000b 7075 6164 6574 0000 0000 0000 0000        // update
0000040 0113 6574 6d72 6163 0070 0000 0000 0000        // termcap
0000050 00ee 746d 6261 0000 0000 0000 0000 0000        // mtab
0000060 0000 746d 6261 007e 0000 0000 0000 0000        // 空闲，未使用。
0000070 007c 616d 6967 0063 0000 0000 0000 0000        // magic
0000080 0016 7270 666f 6c69 0065 0000 0000 0000        // profile
0000090 007e 6170 7373 6477 0000 0000 0000 0000        // passwd
00000a0 0081 7267 756f 0070 0000 0000 0000 0000        // group
00000b0 01ee 746d 6f6f 736c 0000 0000 0000 0000        // mtools
00000c0
[/]#
```





Demo


inode table
20 21 22 23 24 25 26 27 28 29 30
0 1 2 3 4 5 6 …
Boot Block
Super Block
Root Dir
File1.c
mydir
myfile
myfile
38
39
40
41
42
43
Root dir
File1.c
mydir
myfile
Status : dir
Size : **
Data blocks:
20
Status : file
Size : ***
Data blocks :
21 22 23
Status : dir
Size : **
Data blocks :
24
Status : file
Size : ****
25 26 27 29 30
31 32
33 34 35 36 37
28
2 .
2 ..
3 File1.c
4 mydir
5 myfile
7 mydir2
4 .
2 ..
10 a.hwp
11 b.c
24 Test.c
19 Note.doc
28 一次间接块号
indirect
directory entry

创建360k Minix 1.0 文件系统

```
guojun@ubuntu: ~/myfs
File  Edit  View  Search  Terminal  Help
guojun@ubuntu:~/myfs$ dd if=/dev/zero of=minix.img bs=1k count=360
360+0 records in
360+0 records out
368640 bytes (369 kB, 360 KiB) copied, 0.000803917 s, 459 MB/s
guojun@ubuntu:~/myfs$ mkfs.minix minix.img
128 inodes
360 blocks
Firstdatazone=8 (8)
Zonesize=1024
Maxsize=268966912

guojun@ubuntu:~/myfs$
```

sudo mount minix.img -o loop mnt

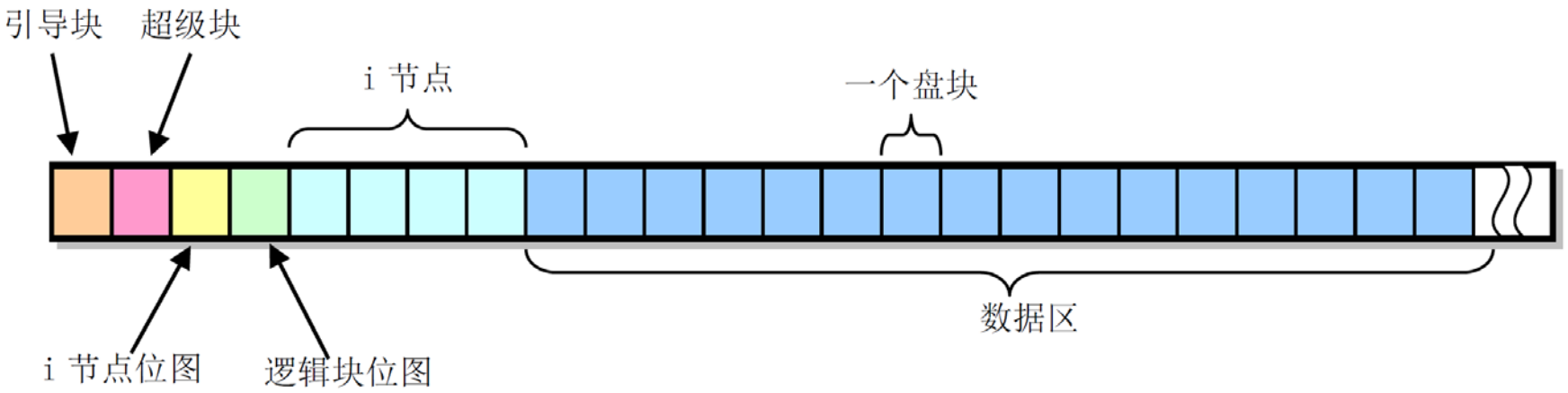
```
guojun@ubuntu: ~/myfs
File  Edit  View  Search  Terminal  Help
guojun@ubuntu:~/myfs$ hexdump -C minix.img
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  80 00 68 01 01 00 01 00  08 00 00 00 00 1c 08 10  |..h.............|
00000410  8f 13 01 00 00 00 00 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  |................|
*
00000800  0b 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  |................|
*
00000c00  43 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  |................|
00000c20  00 00 00 00 00 00 00 00  00 00 00 00 fe ff ff ff  |................|
00000c30  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
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  |................|
*
00002000  01 00 2e 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002020  01 00 2e 2e 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002040  00 00 2e 61 61 61 2e 74  78 74 2e 73 77 70 00 00  |...aaa.txt.swp..|
00002050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002060  03 00 61 61 61 2e 74 78  74 00 00 00 00 00 00 00  |..aaa.txt.......|
00002070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002400  62 30 56 49 4d 20 38 2e  30 00 00 00 00 10 00 00  |b0VIM 8.0.......|
00002410  00 00 00 00 00 00 00 00  87 3a 00 00 67 75 6f 6a  |.........:..guoj|
00002420  75 6e 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |un..............|
00002430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002440  00 00 00 00 75 62 75 6e  74 75 00 00 00 00 00 00  |....ubuntu......|
00002450  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002460  00 00 00 00 00 00 00 00  00 00 00 00 7e 67 75 6f  |............~guo|
00002470  6a 75 6e 2f 6d 79 66 73  2f 6d 6e 74 2f 61 61 61  |jun/myfs/mnt/aaa|
00002480  2e 74 78 74 00 00 00 00  00 00 00 00 00 00 00 00  |.txt............|
00002490  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000027e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 0d 55  |...............U|
000027f0  33 32 31 30 00 00 00 00  23 22 21 20 13 12 55 00  |3210....#"! ..U.|
00002800  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

对比超级块

```
*
00000400  80 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，最大文件长度
        unsigned short s_magic;         //0x138f,minix魔数
};
```

对比i节点位图

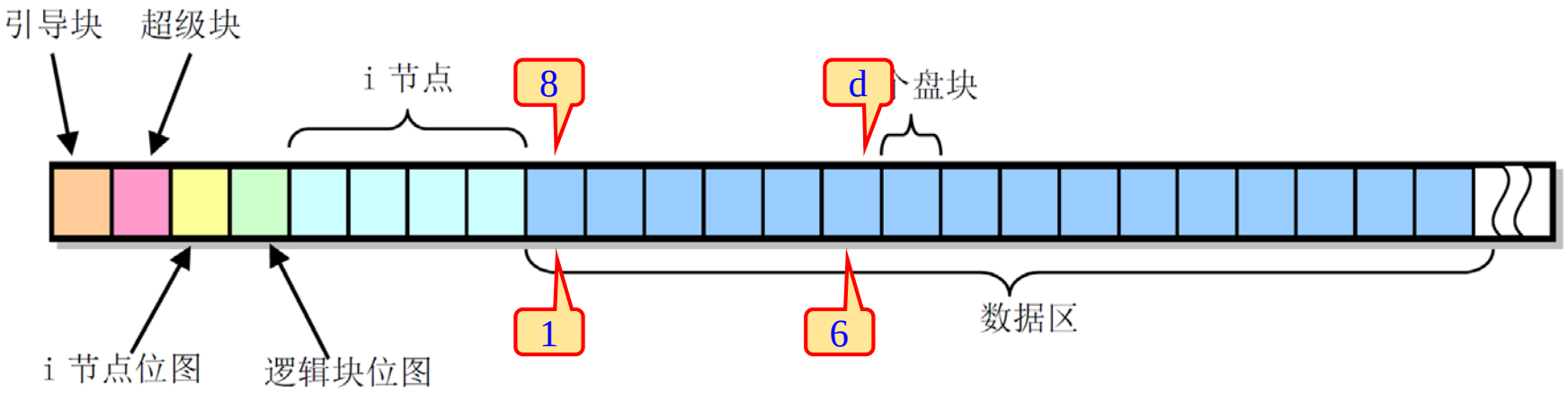


0x0b
0000 1011
1和3在使用

i 节点位图对应 16*8=128 个 i 节点

```
*
00000800  0b 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  |................|
*
```

对比数据区块位图



0x43
0100 0011
1和6在使用

数据区有44*8=352 盘块

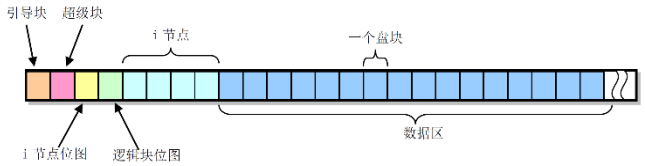
```
*
00000c00  43 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |C...............|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000c20  00 00 00 00 00 00 00 00  00 00 00 00 fe ff ff ff  |................|
00000c30  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff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

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 m_inode {
        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
};
```



对比数据区

```
guojun@ubuntu: ~/myfs/mnt
File  Edit  View  Search  Terminal  Help
guojun@ubuntu:~/myfs/mnt$ ls -ali
total 6
     1 drwxr-xr-x 2 guojun     232  128 Jun 15 23:52 .
655740 drwxr-xr-x 3 guojun guojun 4096 Jun 16 00:18 ..
     3 -rw-r--r-- 1 guojun     232    4 Jun 15 23:52 aaa.txt
guojun@ubuntu:~/myfs/mnt$ cat aaa.txt
aaa
guojun@ubuntu:~/myfs/mnt$
```

8号数据块

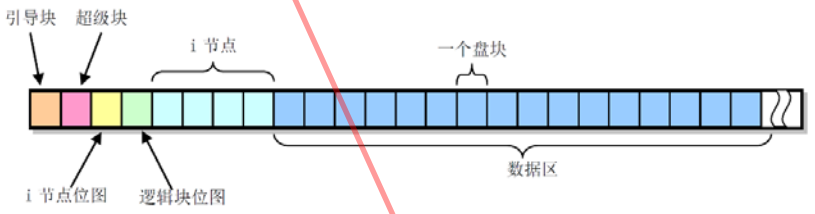
0x01表示i节点号

2e表示.
2e2e表示..

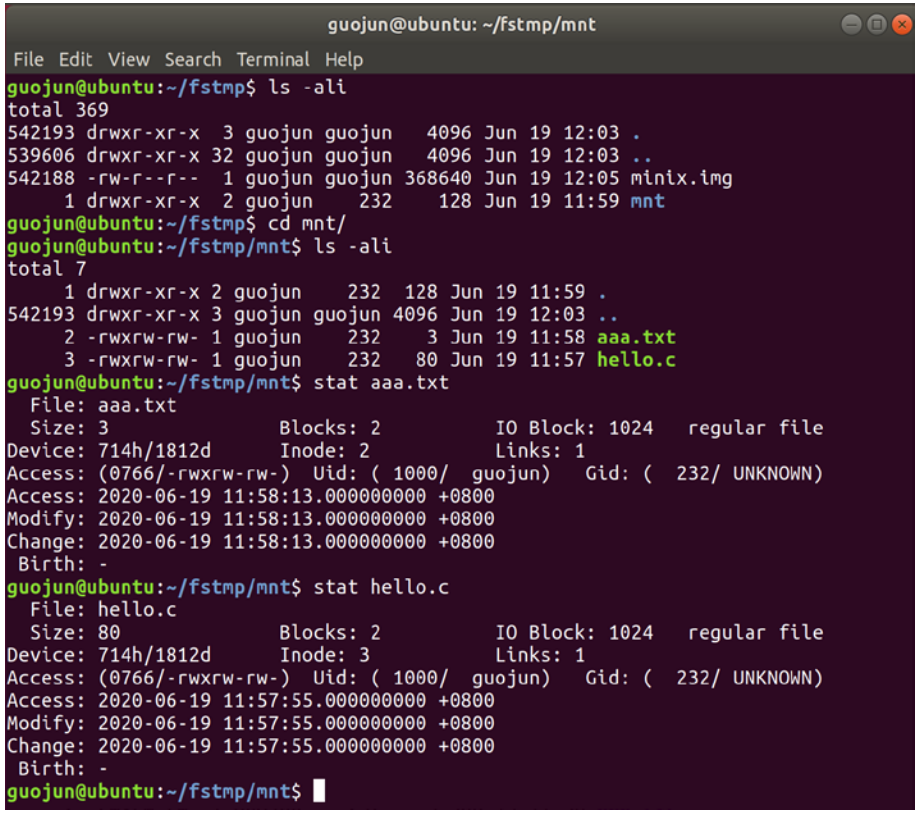
```
*
00002000  01 00 2e 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002020  01 00 2e 2e 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002040  00 00 2e 61 61 61 2e 74  78 74 2e 73 77 70 00 00  |...aaa.txt.swp..|
00002050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002060  03 00 61 61 61 2e 74 78  74 00 00 00 00 00 00 00  |..aaa.txt.......|
00002070  00 00 00 00 00 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
00001050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
00003400  61 61 61 0a 00 00 00 00  00 00 00 00 00 00 00 00  |aaa.............|
00003410  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

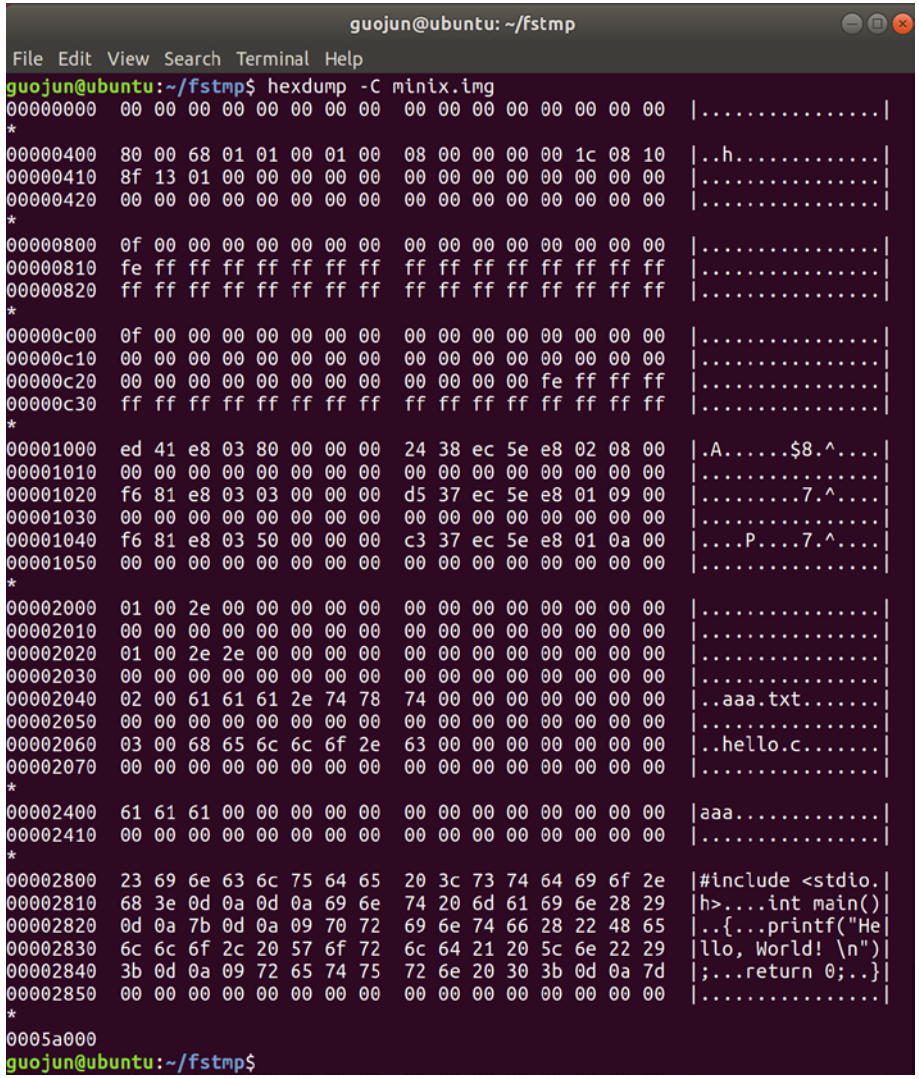
file

0x0d 数据块



Demo 2


guojun@ubuntu: ~/fstmp/mnt
File Edit View Search Terminal Help
guojun@ubuntu:~/fstmp$ ls -ali
total 369
542193 drwxr-xr-x 3 guojun guojun   4096 Jun 19 12:03 .
539606 drwxr-xr-x 32 guojun guojun   4096 Jun 19 12:03 ..
542188 -rw-r--r-- 1 guojun guojun 368640 Jun 19 12:05 minix.img
     1 drwxr-xr-x 2 guojun   232   128 Jun 19 11:59 mnt
guojun@ubuntu:~/fstmp$ cd mnt/
guojun@ubuntu:~/fstmp/mnt$ ls -ali
total 7
     1 drwxr-xr-x 2 guojun   232 128 Jun 19 11:59 .
542193 drwxr-xr-x 3 guojun guojun 4096 Jun 19 12:03 ..
     2 -rwxrw-rw- 1 guojun   232   3 Jun 19 11:58 aaa.txt
     3 -rwxrw-rw- 1 guojun   232   80 Jun 19 11:57 hello.c
guojun@ubuntu:~/fstmp/mnt$ stat aaa.txt
  File: aaa.txt
  Size: 3   Blocks: 2   IO Block: 1024   regular file
Device: 714h/1812d   Inode: 2   Links: 1
Access: (0766/-rwxrw-rw-) Uid: (1000/ guojun)   Gid: (232/ UNKNOWN)
Access: 2020-06-19 11:58:13.000000000 +0800
Modify: 2020-06-19 11:58:13.000000000 +0800
Change: 2020-06-19 11:58:13.000000000 +0800
 Birth: -
guojun@ubuntu:~/fstmp/mnt$ stat hello.c
  File: hello.c
  Size: 80   Blocks: 2   IO Block: 1024   regular file
Device: 714h/1812d   Inode: 3   Links: 1
Access: (0766/-rwxrw-rw-) Uid: (1000/ guojun)   Gid: (232/ UNKNOWN)
Access: 2020-06-19 11:57:55.000000000 +0800
Modify: 2020-06-19 11:57:55.000000000 +0800
Change: 2020-06-19 11:57:55.000000000 +0800
 Birth: -
guojun@ubuntu:~/fstmp/mnt$


guojun@ubuntu: ~/fstmp
File Edit View Search Terminal Help
guojun@ubuntu:~/fstmp$ hexdump -C minix.img
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00000400 80 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 |................|
*
00000800 0f 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 |................|
*
00000c00 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000c10 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 fe ff ff ff |................|
00000c30 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |................|
*
00001000 ed 41 e8 03 80 00 00 00 24 38 ec 5e e8 02 08 00 |.A......$8.^....|
00001010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00001020 f6 81 e8 03 03 00 00 00 d5 37 ec 5e e8 01 09 00 |.........7.^....|
00001030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00001040 f6 81 e8 03 50 00 00 00 c3 37 ec 5e e8 01 0a 00 |....P....7.^....|
00001050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00002000 01 00 2e 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00002010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00002020 01 00 2e 2e 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00002030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00002040 02 00 61 61 61 2e 74 78 74 00 00 00 00 00 00 00 |..aaa.txt.......|
00002050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00002060 03 00 68 65 6c 6c 6f 2e 63 00 00 00 00 00 00 00 |..hello.c.......|
00002070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00002400 61 61 61 00 00 00 00 00 00 00 00 00 00 00 00 00 |aaa.............|
00002410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00002800 23 69 6e 63 6c 75 64 65 20 3c 73 74 64 69 6f 2e |#include <stdio.|
00002810 68 3e 0d 0a 0d 0a 69 6e 74 20 6d 61 69 6e 28 29 |h>....int main()|
00002820 0d 0a 7b 0d 0a 09 70 72 69 6e 74 66 28 22 48 65 |..{...printf("He|
00002830 6c 6c 6f 2c 20 57 6f 72 6c 64 21 20 5c 6e 22 29 |llo, World! \n")|
00002840 3b 0d 0a 09 72 65 74 75 72 6e 20 30 3b 0d 0a 7d |;...return 0;..}|
00002850 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
0005a000
guojun@ubuntu:~/fstmp$

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 –i 查看对应的inode号