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。

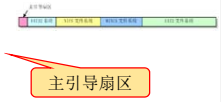
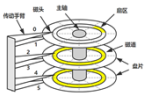


表 9-11 硬盘分区表项结构

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。



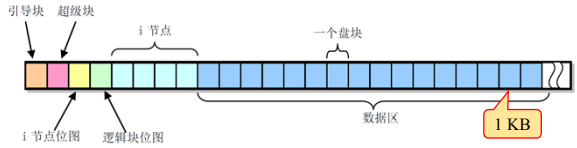
Chapter A4

Minix

Minix 1.0

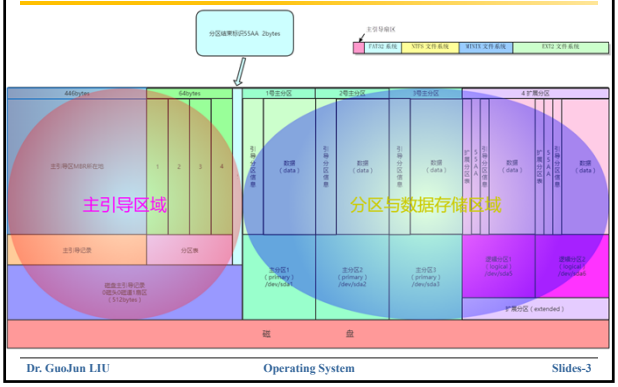
Minix

硬盘设备上的分区和文件系统

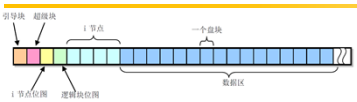


建有 MINIX 文件系统的块设备上各部分的布局示意图

磁盘存储结构逻辑图



MINIX 文件系统的超级块结构




```
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;
    unsigned char s_rd_only;
    unsigned char s_dirt;
} - end super_block - ;
```

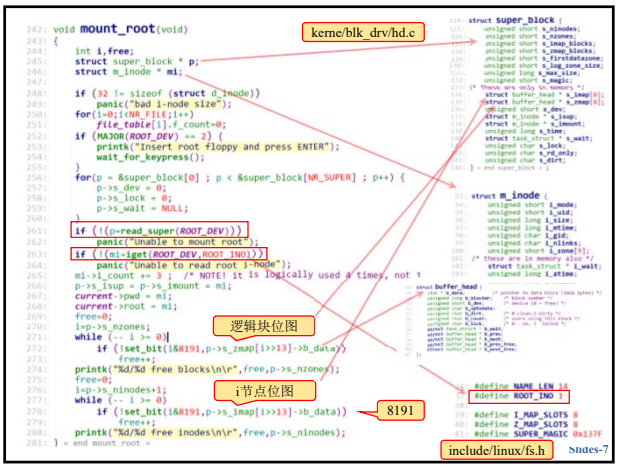
出现在盘上和内存中的字段

仅在内存中使用的字段

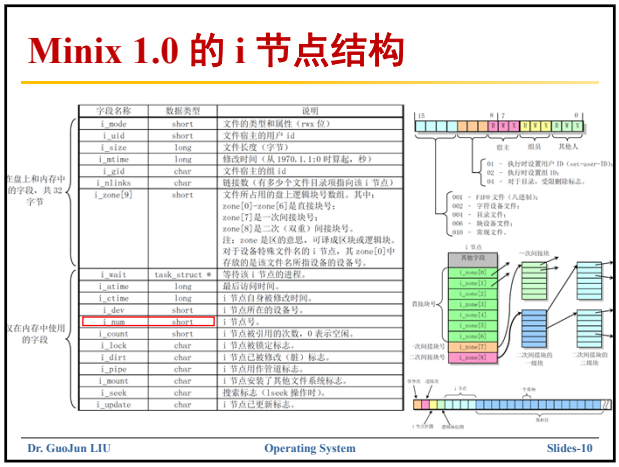
super.c的mount_root()中初始化

super_block [NR_SUPER] 共8项

字段名称	数据类型	说明
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	Log2(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）
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

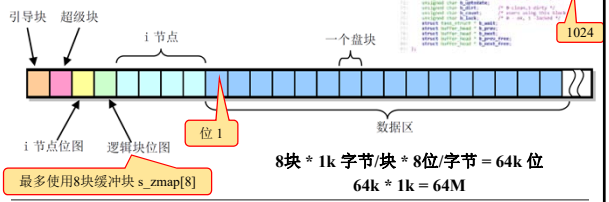


Minix 1.0 的 i 节点结构



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



8块 * 1k 字节/块 * 8位/字节 = 64k 位

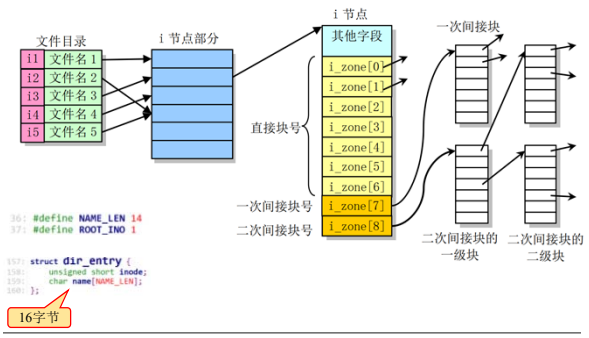
64k * 1k = 64M

Minix 1.0 的 i 节点结构

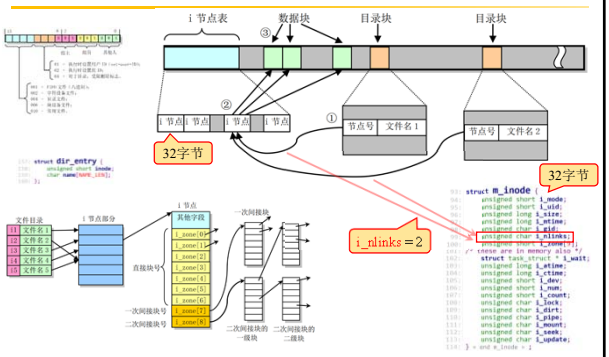
i 节点位图和 i 节点

- 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

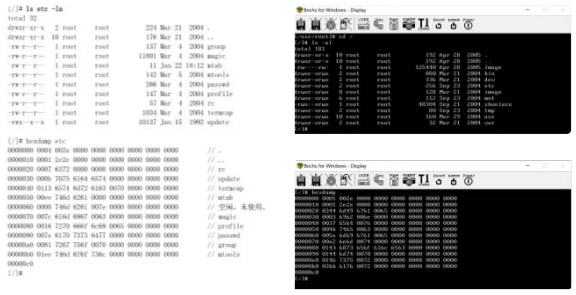
通过文件名找对应文件磁盘块位置



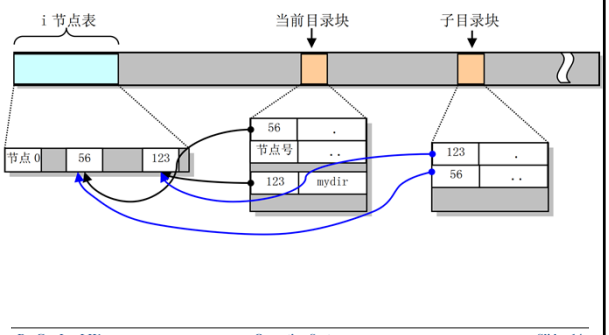
从文件名获取其数据块



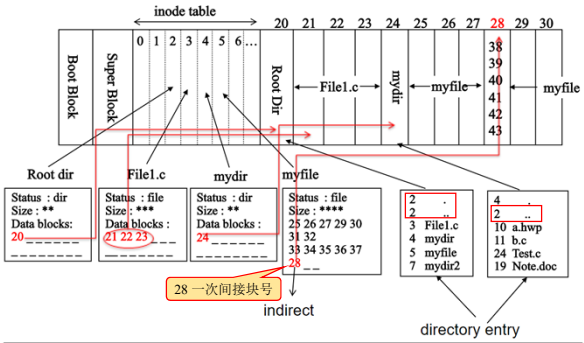
hexdump 查看目录内容

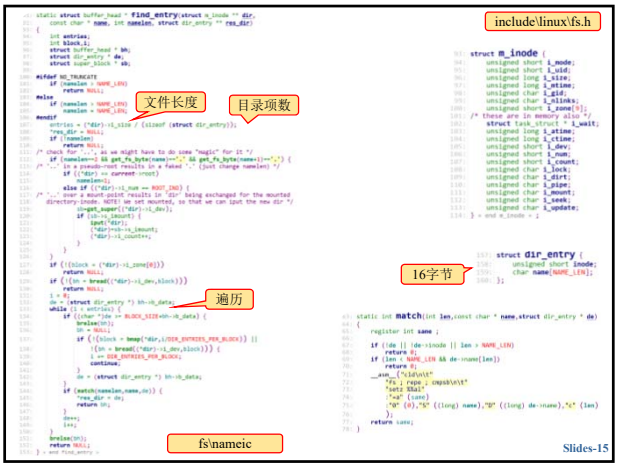


目录文件目录项和子目录链接

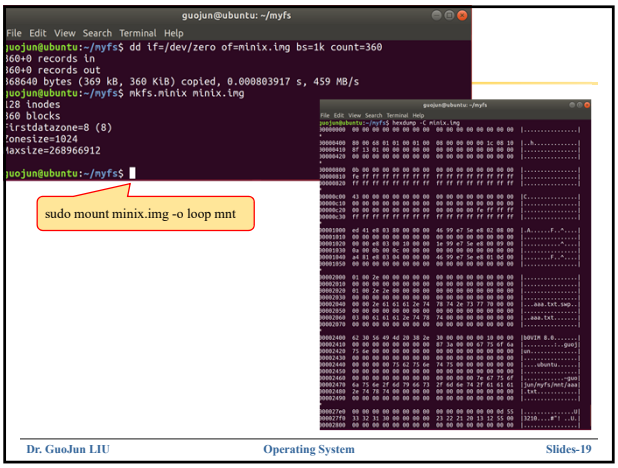


Demo

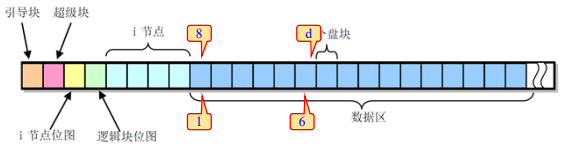




创建360k Minix 1.0 文件系统



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
};
```

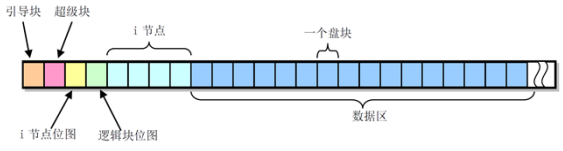

对比i节点数据

```
*
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

8号数据块

0x01表示i节点号

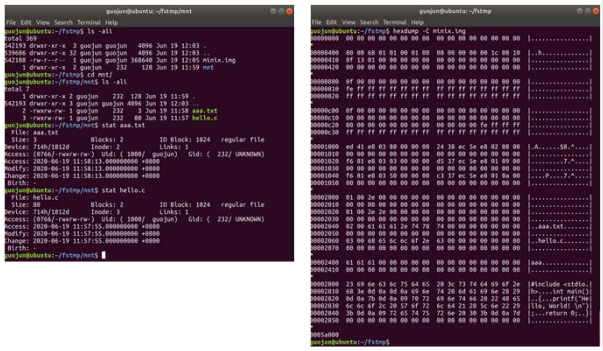
2e表示.
2e2e表示..

```
*
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  |................|
*
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



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