

## 36 | Linux文件系统（二）：Linux如何存放文件？

2022-10-24 LMOS 来自北京

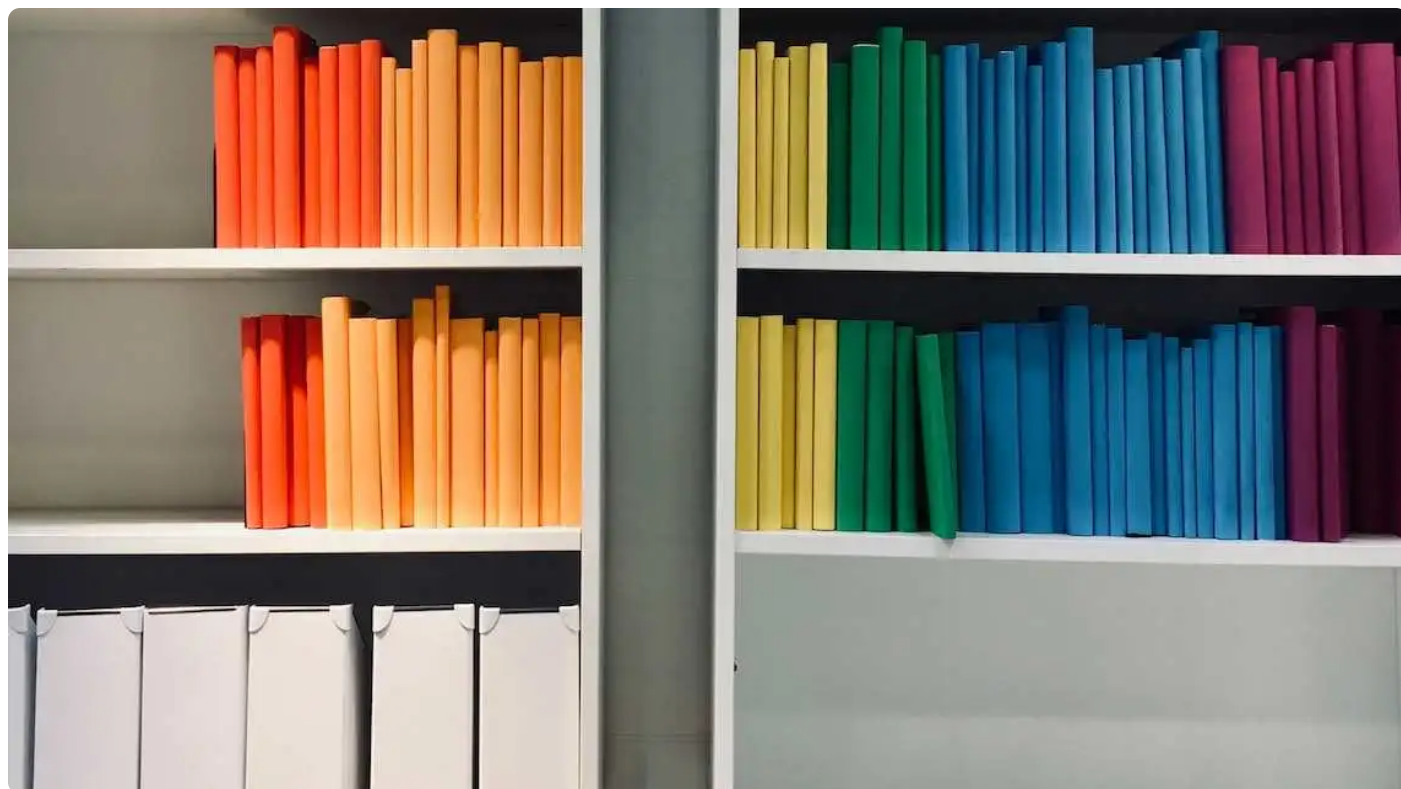


天下无鱼

<https://shikey.com/>

《计算机基础实战课》

[课程介绍 >](#)



讲述：陈晨

时长 08:26 大小 7.70M



你好，我是 LMOS。

通过上节课的学习，我们已经对 Ext3 文件系统的结构非常了解了。这种了解究竟正确与否，还是需要通过写代码来验证。这节课我会带你读取 Ext3 文件系统中的文件，帮你加深对 Ext3 的理解。

我假定你已经学会了怎么建立一个虚拟硬盘并将其格式化为 Ext3 文件系统。如果记不清了，请回到 [上节课](#) 复习一下。课程的配套代码，你需要从 [这里](#) 下载。

### 打开虚拟硬盘

想要从虚拟硬盘读取文件，首先要做的当然是打开虚拟硬盘。但我们的硬盘是个文件，所以这就变成了打开了一个文件，然后对文件进行读写就行。这些操作我们已经非常熟悉了，不过多展开。

这次我们不用 `read` 命令来读取虚拟硬盘文件数据，因为那样做还需要处理分配临时内容和文件定位的问题，操作比较繁琐。这里我们直接用 `mmap` 将整个文件映射到虚拟文件中，这样就能像访问内存一样很方便地访问文件了。

下面我们首先实现 `mmap` 映射读取文件这个功能，代码如下所示：

 复制代码

```
1  int init_in_hdfile()
2  {
3      struct stat filestat;
4      size_t len = 0;
5      void* buf = NULL;
6      int fd = -1;
7      // 打开虚拟硬盘文件
8      fd = open("./hd.img", O_RDWR, S_IRWXU|S_IRWXG|S_IRWXO);
9      if(fd < 0)
10     {
11         printf("打开文件失败\n");
12         return -1;
13     }
14     // 获取文件信息，比如文件大小
15     fstat(fd, &filestat);
16     // 获取文件大小
17     len = filestat.st_size;
18     // 映射整个文件到进程的虚拟内存中
19     buf = mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
20     if(buf == NULL)
21     {
22         printf("映射文件失败\n");
23         return -2;
24     }
25     // 保存地址 长度大小 文件句柄 到全局变量
26     hdaddr = buf;
27     hdsiz = len;
28     hdfild = fd;
29     return 0;
30 }
```

我们把打开硬盘文件以及将其映射到进程的虚拟内存中的功能，封装在 `init_in_hdfile` 函数中，并把映射返回的地址、文件长度、文件句柄保存到全局变量中，以便后面使用。

## 获取 Ext3 文件系统超级块

好，作为硬盘的文件已经完成映射，下面我们就来获取其中的 `Ext3` 文件系统超级块。

**Ext3** 文件系统超级块固定存放在硬盘 2 号扇区的开始地址，硬盘扇区从 0 开始计数。我们需要把扇区号转换成文件中对应的偏移量，然后把这个偏移量转换成文件映射虚拟内存中的地址，才能访问到正确的数据。

下面我们开始写代码，如下所示：

 复制代码

```
1 // 将扇区号转换成文件映射的虚拟内存地址
2 void* sector_to_addr(__u64 nr)
3 {
4     return (void*)((__u64)hdaddr + (nr * SECTOR_SIZE));
5 }
6 // 将储存块号转换成文件映射的虚拟内存地址
7 void* block_to_addr(__u64 nr)
8 {
9     return (void*)((__u64)hdaddr + (nr * block_size));
10 }
11 // 获取超级块的地址
12 struct ext3_super_block* get_superblock()
13 {
14     return (struct ext3_super_block*)sector_to_addr(2);
15 }
```

**Ext3** 的超级块结构，定义在工程目录下的 `ext3fs.h` 头文件中。代码的 `get_superblock` 函数中正是通过 `sector_to_addr` 函数对第二号扇区做了转换，之后还加上了映射文件的首地址，才能访问硬盘文件中的超级块。

我们可以调用 `dump_super_block` 函数，打印超级块的一些信息，如下图所示：

```
imos@imos-PC: ~/计算机基础/code/33$ make run
inode节点总数:25688
储存块总数:182400
空闲inode节点总数:25677
空闲储存块总数:93487
第一个inode节点号:11
第一个储存块:1
储存块大小:1024
每组包含inode节点数:1976
每组包含储存块数:8192
最后写入时间:1659665215秒
最后挂载时间:1659665215秒
每个inode节点大小:128
imos@imos-PC: ~/计算机基础/code/33$
```

从上面的截图，我们能知道文件系统的全局信息，也就是该文件系统有多少个储存块、inode、储存块大小，每个块组多少个储存块等相关信息。

## 获取 Ext3 文件系统块组描述符表

我们知道，Ext3 文件系统将硬盘分区划分成一个个块组，在超级块的下一个储存块中保存着块组描述符表。如果超级块在 0 号储存块中，块组描述符表就是 1 号储存块中；如果超级块在 1 号储存块，块组描述符表就在 2 号储存块中。

一个块组中有储存块位图块，有 inode 节点位图块，也有 inode 节点表。要获取 Ext3 文件系统块组描述符表，我们只要知道它所在的储存块，就能读取其中的信息。

下面我们代码实现这一步：

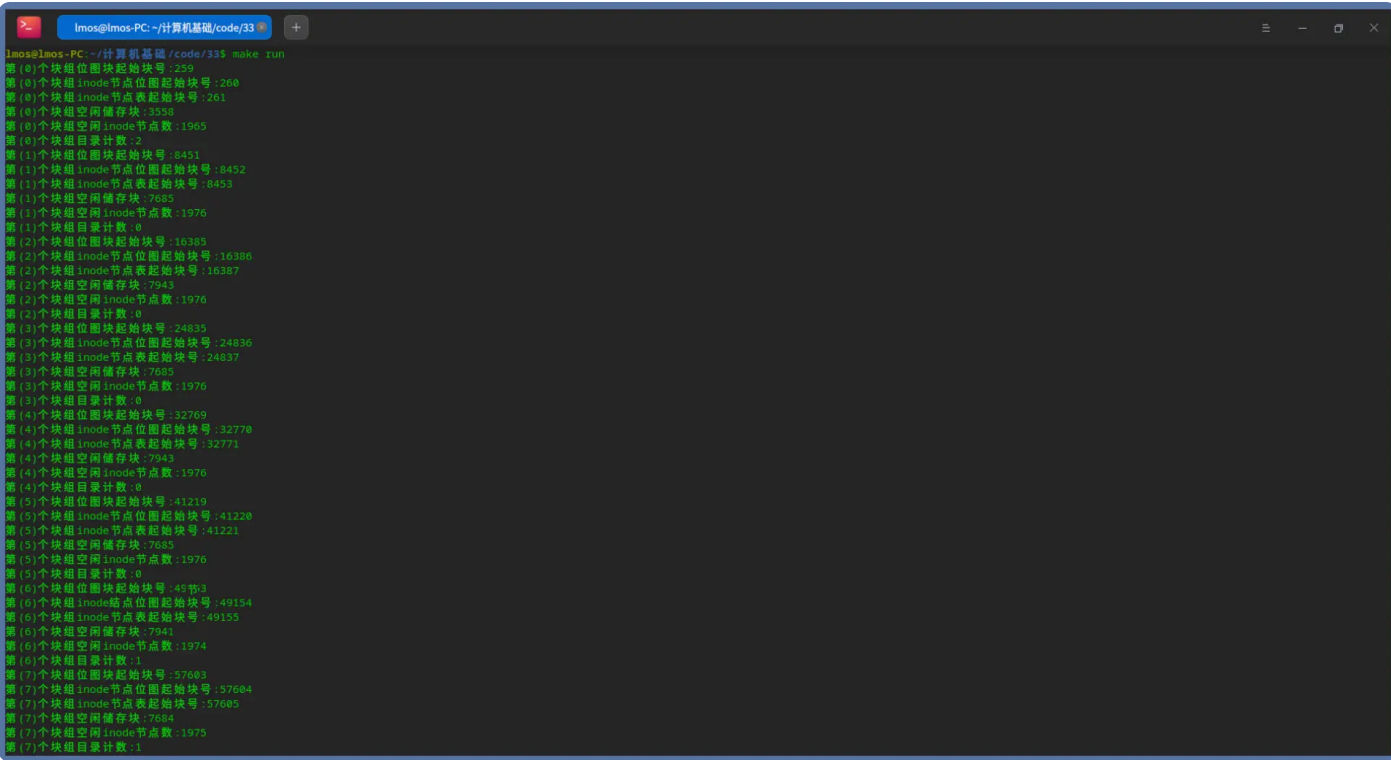
 复制代码

```
1 void get_group_table(struct ext3_group_desc** outgtable, int* outnr )
2 {
3     // 计算总块组数
4     int gnr = super->s_blocks_count / super->s_blocks_per_group;
5     // 获取块组描述表的首地址
6     struct ext3_group_desc* group = (struct ext3_group_desc* ) block_to_addr(2);
7     *outgtable = group;
8     *outnr = gnr;
9     return;
10 }
```

以上获取块组描述符表的函数，我们可以通过参数，返回两个块组描述符表的首地址和个数。

这里我已经为你写好了 `dump_all_group` 函数，你只要调用它，就可以直接获取块组描述符表信息了。

接下来我们看看打印出来的块组描述符表信息，如下所示：



```
imos@imos-PC: ~/计算机基础/code/33 $ make run
第(0)个块组位图块起始块号: 259
第(0)个块组inode节点位图起始块号: 260
第(0)个块组inode节点表起始块号: 261
第(0)个块组空闲缓存块: 3558
第(0)个块组空闲inode节点数: 1965
第(0)个块组目录计数: 2
第(1)个块组位图块起始块号: 8451
第(1)个块组inode节点位图起始块号: 8452
第(1)个块组inode节点表起始块号: 8453
第(1)个块组空闲缓存块: 7685
第(1)个块组空闲inode节点数: 1976
第(1)个块组目录计数: 0
第(2)个块组位图块起始块号: 16385
第(2)个块组inode节点位图起始块号: 16386
第(2)个块组inode节点表起始块号: 16387
第(2)个块组空闲缓存块: 7945
第(2)个块组空闲inode节点数: 1976
第(2)个块组目录计数: 0
第(3)个块组位图块起始块号: 24835
第(3)个块组inode节点位图起始块号: 24836
第(3)个块组inode节点表起始块号: 24837
第(3)个块组空闲缓存块: 7685
第(3)个块组空闲inode节点数: 1976
第(3)个块组目录计数: 0
第(4)个块组位图块起始块号: 32769
第(4)个块组inode节点位图起始块号: 32770
第(4)个块组inode节点表起始块号: 32771
第(4)个块组空闲缓存块: 7945
第(4)个块组空闲inode节点数: 1976
第(4)个块组目录计数: 0
第(5)个块组位图块起始块号: 41219
第(5)个块组inode节点位图起始块号: 41220
第(5)个块组inode节点表起始块号: 41221
第(5)个块组空闲缓存块: 7685
第(5)个块组空闲inode节点数: 1976
第(5)个块组目录计数: 0
第(6)个块组位图块起始块号: 49154
第(6)个块组inode节点位图起始块号: 49155
第(6)个块组空闲缓存块: 7941
第(6)个块组空闲inode节点数: 1974
第(6)个块组目录计数: 1
第(7)个块组位图块起始块号: 57603
第(7)个块组inode节点位图起始块号: 57604
第(7)个块组inode节点表起始块号: 57605
第(7)个块组空闲缓存块: 7684
第(7)个块组空闲inode节点数: 1975
第(7)个块组目录计数: 1
```

## 获取 Ext3 文件系统根目录

要想在文件系统中读取文件，就必须从其根目录开始，一层一层查找，直到找到文件的 `inode` 节点。

可是根目录在哪里呢？它就在第一个块组中 `inode` 节点表中的第 2 个 `inode`，也就是根目录的 `inode` 节点，这个 `inode` 节点对应的数据块中储存的目录项数据。目录项可以指向一个目录，也可以指向一个文件，就这样一层层将目录或者文件组织起来了。

下面我们就来写代码实现这一步，如下所示：

 复制代码

```
1 // 获取根目录的inode的地址
```

```

2 struct ext3_inode* get_rootinode()
3 {
4     // 获取第1个块组描述符
5     struct ext3_group_desc* group = (struct ext3_group_desc* ) block_to_addr(2);
6     // 获取该块组的inode表的块号
7     __u32 ino = group->bg_inode_table;
8     // 获取第二个inode
9     struct ext3_inode* inp = (struct ext3_inode* )((__u64)block_to_addr(ino)+(sup
10     return inp;
11 }
12 // 获取根目录的开始的数据项的地址
13 struct ext3_dir_entry* get_rootdir()
14 {
15     // 获取根目录的inode
16     struct ext3_inode* inp = get_rootinode();
17     // 返回根目录的inode中第一个数据块的地址，就是根目录的数据
18     return (struct ext3_dir_entry*)block_to_addr(inp->i_block[0]);
19 }

```

上面代码中有两个函数，一个是获取根目录 `inode` 的地址，有了它才能获取根目录的数据，由于我们的文件系统没有太多目录和文件，所以只用一块储存块就能放下所有的目录项目。

我已经为你写好了代码，用于显示根目录下所有的目录和文件，现在你只要调用 `dump_dirs` 函数可以了，如下所示：

```

imos@imos-PC: ~/计算机基础/code/33$ make run
目录项对应的inode节点:2
目录项长度:12
目录项名称长度:1
目录项类型:2, 目录
目录项名称:
目录项对应的inode节点:2
目录项长度:12
目录项名称长度:2
目录项类型:2, 目录
目录项名称:..
目录项对应的inode节点:11
目录项长度:44
目录项名称长度:10
目录项类型:2, 目录
目录项名称:lost+found
目录项对应的inode节点:13857
目录项长度:10
目录项名称长度:6
目录项类型:2, 目录
目录项名称:ext3fs
目录项对应的inode节点:13833
目录项长度:940
目录项名称长度:4
目录项类型:2, 目录
目录项名称:info
imos@imos-PC: ~/计算机基础/code/33$

```

由上可知，根目录下有 5 个子目录，分别是：`.`、`..`、`lost+found`、`ext3fs`、`info`。`ext3fs` 和 `info` 是我主动建立的，用于测试。我还在 `ext3fs` 目录下建立了一个 `ext3.txt` 文件，并在其中写入了

“Hello EXT3 File System!!”数据，下面我们就去读取它的文件数据。

## 获取 Ext3 文件系统文件

现在我们要读取 Ext3 文件系统下的 `/ext3fs/ext3.txt` 文件，但是我们必须要从根目录开始，查找 `ext3fs` 目录对应 `inode` 节点。然后在 `ext3fs` 目录数据中，找到 `ext3.txt` 文件对应的 `inode` 节点，读取该 `inode` 中直接或者间接地址块中块号对应的储存块，那里就是文件的真实数据。

目前我们已经能读取根目录的数据了，只要再操作两步，就可以查到 `ext3.txt` 对应的 `inode`。

下面我们开始写代码，如下所示：

 复制代码

```
1 // 判定文件和目录
2 struct ext3_dir_entry* dir_file_is_ok(struct ext3_dir_entry* dire, __u8 type, c
3 {
4     // 比较文件和目录类型和名称
5     if(dire->file_type == type)
6     {
7         if(0 == strncmp(name, dire->name, dire->name_len))
8         {
9             return dire;
10        }
11    }
12    return NULL;
13 }
14 // 查找一个块中的目录项
15 struct ext3_dir_entry* find_dirs_on_block(void* blk, size_t size, __u8 type, ch
16 {
17     struct ext3_dir_entry* dire = NULL;
18     void* end = (void*)((__u64)blk + size);
19     for (void* dir = blk; dir < end;)
20     {
21         // 判定是否找到
22         dire = dir_file_is_ok((struct ext3_dir_entry*)dir, type, name);
23         if(dire != NULL)
24         {
25             return dire;
26         }
27         // 获取下一个目录项地址
28         dir = get_next_dir_addr((struct ext3_dir_entry*)dir);
29     }
30     return NULL;
31 }
32 // 在一个目录文件中查找目录或者文件
33 struct ext3_dir_entry* find_dirs(struct ext3_inode* inode, __u8 type, char* na
```

```

34 {
35     struct ext3_dir_entry* dir = NULL;
36     __s64 filesize = inode->i_size;
37     // 查找每个直接块
38     for (int i = 0; (i < (EXT3_N_BLOCKS - 3))&&(filesize > 0); i++, filesize -= (
39     {
40         // 查找一个储存块
41         dir = find_dirs_on_block(block_to_addr(inode->i_block[i]), (size_t)filesize
42         if(dir != NULL)
43         {
44             return dir;
45         }
46     }
47     return NULL;
48 }

```

上述代码中的三个函数的作用就是查找我们需要的目录和文件。具体是这样的：`find_dirs` 用来查找整个 `inode`；`find_dirs_on_block` 用来查找 `inode` 中一个储存块；`dir_file_is_ok` 用于判定每个查找到的目录项，如果找到就返回对应的地址，否则返回 `NULL`。

下面我们在 `read_file` 函数中调用上述函数，如下所示：

 复制代码

```

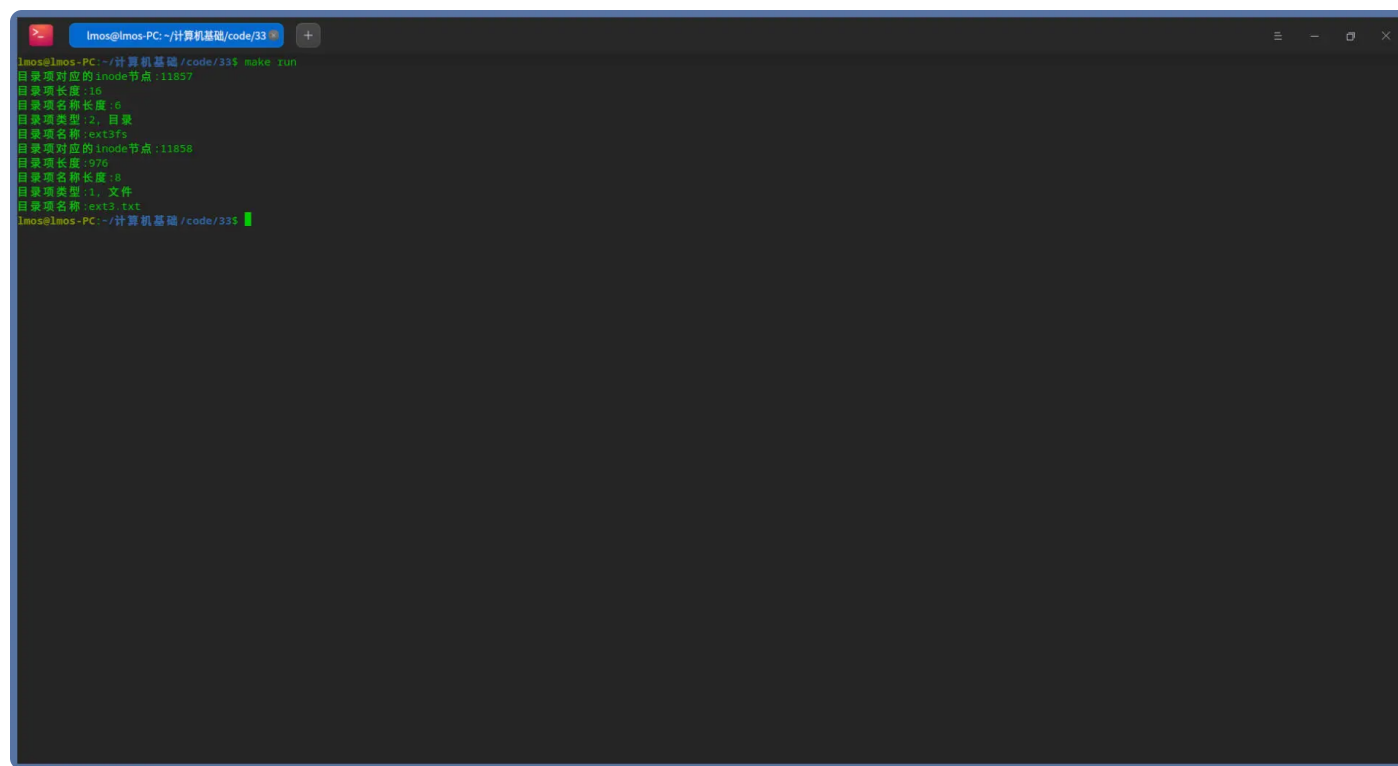
1 void read_file()
2 {
3     struct ext3_dir_entry* dir = NULL;
4     // 查找ext3fs目录
5     dir = find_dirs(rootinode, 2, "ext3fs");
6     if(dir == NULL)
7     {
8         printf("没有找到ext3fs目录\n");
9         return;
10    }
11    // 显示ext3fs目录的目录项信息
12    dump_one_dir(dir);
13    // 查找ext3fs目录下的ext3.txt文件
14    dir = find_dirs(get_inode(dir->inode), 1, "ext3.txt");
15    if(dir == NULL)
16    {
17        printf("没有找到ext3.txt\n");
18        return;
19    }
20    // 显示ext3.txt文件的目录项信息
21    dump_one_dir(dir);
22    return;
23 }

```



以上代码的作用是这样的：第一步查找 `ext3fs` 目录，第二步查找 `ext3fs` 目录下的 `ext3.txt` 文件，并把它们相应的信息显示出来。

我们把程序运行一下，如下所示：



```
imos@imos-PC: ~/计算机基础/code/33 %  
imos@imos-PC:~/计算机基础/code/33$ make run  
目录项对应的inode节点:11857  
目录项长度:16  
目录项名称长度:6  
目录项类型:2, 目录  
目录项名称:ext3fs  
目录项对应的inode节点:11858  
目录项长度:976  
目录项名称长度:8  
目录项类型:1, 文件  
目录项名称:ext3.txt  
imos@imos-PC:~/计算机基础/code/33$
```

上图中已经显示了 `ext3.txt` 文件的 `inode` 号，根据这个 `inode` 号，我们就能找到对应 `inode` 节点，下面我们进一步写代码读取文件中的数据。代码如下所示：

 复制代码

```
1 void dump_inode_data(struct ext3_inode* inode)  
2 {  
3     // 获取文件大小  
4     __s64 filesize = inode->i_size;  
5     printf("-----\n");  
6     // 展示文件inode的元信息  
7     dump_inode(inode);  
8     printf("-----\n");  
9     for (int i = 0; (i < (EXT3_N_BLOCKS - 3))&&(filesize > 0); i++, filesize -= (  
10 {  
11     // 读取并打印每个储存块中数据内部  
12     printf("%s\n", (char*)block_to_addr(inode->i_block[i]));  
13 }  
14     return;  
15 }  
16  
17 void read_file()  
18 {
```

```

19 struct ext3_dir_entry* dir = NULL;
20 // 查找ext3fs目录
21 dir = find_dirs(rootinode, 2, "ext3fs");
22 if(dir == NULL)
23 {
24     printf("没有找到ext3fs目录\n");
25     return;
26 }
27 // 显示ext3fs目录的目录项信息
28 dump_one_dir(dir);
29 // 查找ext3fs目录下的ext3.txt文件
30 dir = find_dirs(get_inode(dir->inode), 1, "ext3.txt");
31 if(dir == NULL)
32 {
33     printf("没有找到ext3.txt\n");
34     return;
35 }
36 // 显示ext3.txt文件的目录项信息
37 dump_one_dir(dir);
38 // 显示ext3.txt文件的内容信息
39 dump_inode_data(get_inode(dir->inode));
40 return;
41 }

```

在上面的 `dump_inode_data` 函数中，我之所以能用 `printf` 打印文件内存，是因为我清楚 `ext3.txt` 文件存放写入的是文本数据。如果是其它别的数据就不能这样做了。

除了打印文件内容，我们还展示了文件元信息。让我们运行一下，看看结果：

```

lmos@lmos-PC: ~/计算机基础/code/33$ make run
目录项对应的inode结点:11857
目录项长度:16
目录项名称长度:6
目录项类型:2, 目录
目录项名称:ext3fs
目录项对应的inode结点:11858
目录项长度:976
目录项名称长度:0
目录项类型:1, 文件
目录项名称:ext3.txt
-----
文件权限模式:33188
文件所属用户:1000
文件所属用户组:1000
文件大小:24
文件占用的存储块数:2
文件的链接数:1
文件访问时间(秒):1659605483
文件建立时间(秒):1659605483
文件修改时间(秒):1659605483
文件第0个存储块号:51203
文件第1个存储块号:0
文件第2个存储块号:0
文件第3个存储块号:0
文件第4个存储块号:0
文件第5个存储块号:0
文件第6个存储块号:0
文件第7个存储块号:0
文件第8个存储块号:0
文件第9个存储块号:0
文件第10个存储块号:0
文件第11个存储块号:0
文件第12个存储块号:0
文件第13个存储块号:0
文件第14个存储块号:0
-----
Hello, EXT3 File System!
lmos@lmos-PC: ~/计算机基础/code/33$

```

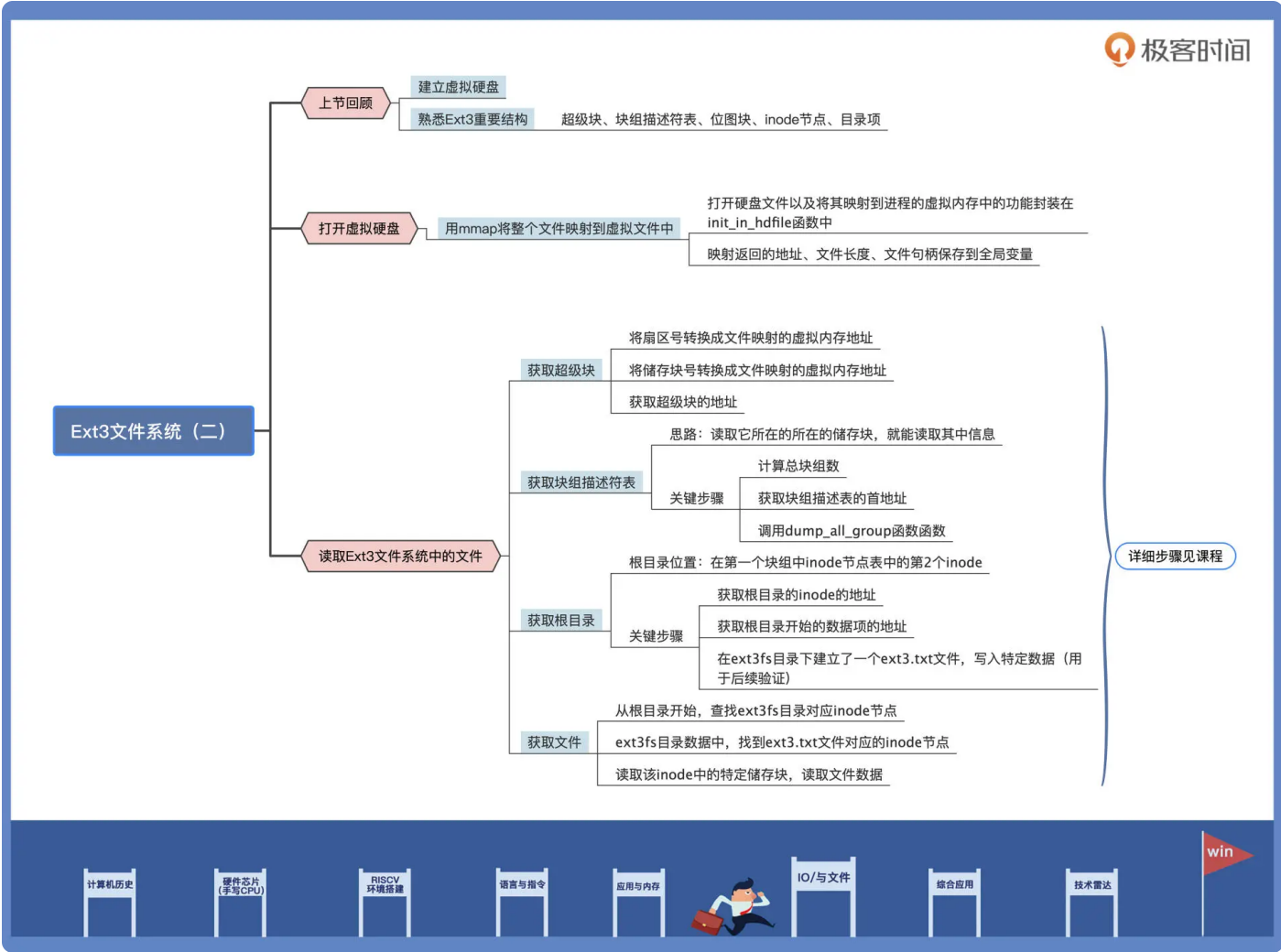
从上图，我们已经清楚地看到文件大小、创建时间、所属用户、占用哪个储存块，最后还打印出了文件的内容——Hello EXT3 File System!!，这与我们之前写入的数据分毫不差。到这里，我们已经验证了 Ext3 文件系统结构，也完成了读文件信息的各类实践。

## 重点回顾

只要认真学完这两节课，我相信你对 Ext3 文件系统已经有了更深入的了解，硬件上的数据修改是完全可以做到的，成为数据修复大师也指日可待。不过，不能利用这些知识去干坏事哦。

今天，为了验证上节课学到的一系列 Ext3 结构，我们通过写代码的方式，在文件系统中读取了文件数据。我们通过获取超级块、块组的描述符表，一步步完整地把文件内容读取出来，打印在屏幕上。对比之下，这正好跟我们先前输入的内容是一样的，也就验证了 Ext3 文件系统结构。

这节课的导图如下所示，供你参考：




## 思考题

请问 **inode** 号 是对应于硬盘分区全局 还是相对于块组的？

进入下个章节之前，希望你可以留言说说学习的感受，或者向我提问。如果觉得课程还不错，别忘了分享给身边更多朋友。

分享给需要的人，Ta购买本课程，你将得 **20** 元

 生成海报并分享

 赞 2  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 35 | Linux文件系统（一）：Linux如何存放文件？

下一篇 37 | 浏览器原理（一）：浏览器为什么要用多进程模型？

## 精选留言 (3)

 写留言



**LockedX**

2022-10-26 来自湖北

可以通过**inode**节点来恢复数据，**inode**节点在发生变化的时候会记录在日志文件中，如果存储改文件的快还没有被覆盖，就可以通过日志文件来恢复**inode**节点这样文件就恢复了。老师放心，我比较老实不会去做坏事的，嘿嘿.....

作者回复: 66666



 1



**TableBear** 

2022-10-24 来自湖北

有几个疑问想请教一下老师：

1. 根目录的目录项存放在**inode**节点列表的第二个**inode**这是规范吗？第一个**inode**存放什么呢？
2. 如果目录项个数超过一个**inode**能表示的范围是不是像数据节点**inode**那样使用一级间接存储块、二级间接存储卡以及三级呢？

作者回复: 文件系统规定的



1



苏流郁宓

2022-10-24 来自湖北

inode相对于块组的啊

作者回复: 嗯嗯



1