

Linux内核源代码导读



中国科学技术大学计算机系
陈香兰 (0551 - 3606864)

xlanchen@ustc.edu.cn

Spring 2009



嵌入式系统实验室

EMBEDDED SYSTEM LABORATORY
SUZHOU INSTITUTE FOR ADVANCED STUDY OF USTC

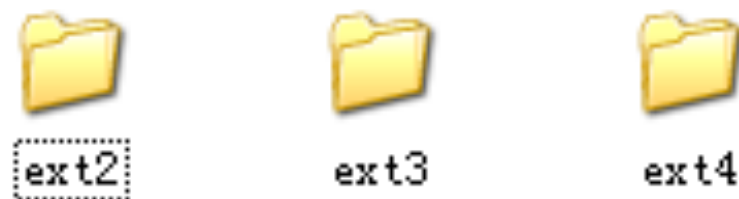
Ext2文件系统简介



嵌入式系统实验室

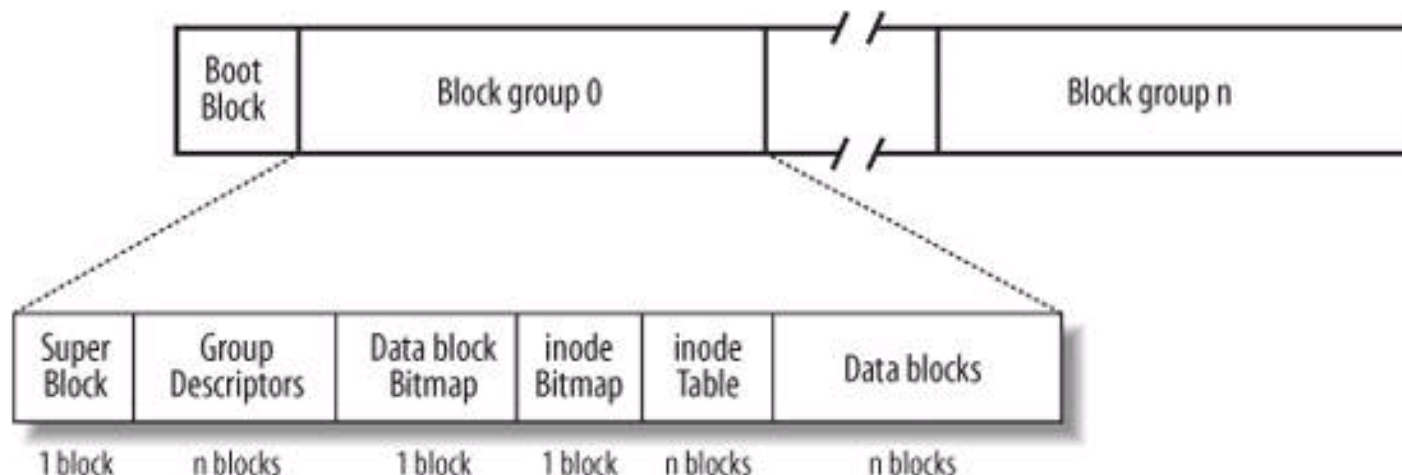
EMBEDDED SYSTEM LABORATORY
SUZHOU INSTITUTE FOR ADVANCED STUDY OF USTC

- ❖ EXT2文件系统是EXT文件系统的升级，在Linux中得到了广泛的使用。



- ❖ 介绍EXT2文件系统的
 - 磁盘组织
 - 目录项和 supported 的文件类型

Layouts of an Ext2 partition and of an Ext2 block group



◆ EXT2磁盘块组中的磁盘块按顺序被组织成：

- 一个用作**超级块**的磁盘块。
 - 在这个磁盘块里，存放了文件系统超级块的一个拷贝；
- N个记录**组描述符**的磁盘块；
- 1个记录**数据块位图**的磁盘块；
- 1个记录**索引结点位图**的磁盘块；
- N个用作**索引结点表**的磁盘块；
- N个用作**数据块**的磁盘块。

冗余，使用块组0
e2fsck更新或恢复
块组大小主要受限于块位图

EXT2的超级块

- ❖ 每个块组的第一个磁盘块用来保存所在EXT2 fs的超级块
- ❖ 多个块组中的超级块形成冗余
 - 在某个或少数几个超级块被破坏时，可用于恢复被破坏的超级块信息。（e2fsck）
- ❖ 注意：大多数数据结构存在两个版本
 - 磁盘存储版本，例如ext2_super_block（阅读）
 - 内存版本，例如ext2_sb_info（阅读）



组描述符

- ❖ 组描述符用来描述一个磁盘块组的相关信息
- ❖ 数据结构为ext2_group_desc（阅读）



索引结点

- ❖ EXT2中所有的索引结点大小相同，都是128个字节。
- ❖ 数据结构
 - 磁盘存储数据结构ext2_inode（阅读）
 - 内存中结构ext2_inode_info（阅读）
- ❖ 理论基础：文件数据块的组织方式
 - 链式（显式 vs 隐式）
 - 索引方式（直接索引，一级索引，二级索引，等等；组合索引）



关于索引节点中的i_block[]

❖ ext2的索引结点中使用了组合索引方式。

```
00241:      __le32      i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */
```

```
00159: /*
00160:  * Constants relative to the data blocks
00161:  */
00162: #define EXT2_NDIR_BLOCKS      12
00163: #define EXT2_IND_BLOCK      EXT2_NDIR_BLOCKS
00164: #define EXT2_DIND_BLOCK      (EXT2_IND_BLOCK + 1)
00165: #define EXT2_TIND_BLOCK      (EXT2_DIND_BLOCK + 1)
00166: #define EXT2_N_BLOCKS      (EXT2_TIND_BLOCK + 1)
```

- 前12项用作直接索引
- 第13项用作间接索引
- 第14项用作二次间接索引
- 第15项用作三次间接索引



索引节点表

- ❖ EXT2的一个磁盘块组中的索引结点存储在一组连续的磁盘块中，形成一个索引结点表。
- ❖ 这组磁盘块中的第一个磁盘块的块号存储在超级块的bg_inode_table数据项中。
- ❖ 根据磁盘块的大小，可以计算出每个磁盘块能容纳多少个索引结点
- ❖ 根据索引结点的总个数，可以计算出索引结点表所需要占用的磁盘块的个数。



数据块位图和索引结点块位图

- ❖ EXT2的空闲盘块分配算法采用了位图法
- ❖ 位图：
便于查找数据块或索引结点的分配信息
- ❖ 每个位（bit）都对应了一个磁盘块：
 - 0，表示对应的磁盘块（或索引结点）空闲
 - 1，表示占用。
- ❖ 2个位图分别占用一个专门的磁盘块。
- ❖ 根据磁盘块的大小，可以计算出每个块组中最多能容纳的数据块个数和索引节点块个数。



(二) EXT2中的目录项和文件类型

- ❖ 在EXT2中，目录是一种特殊的文件，这种文件的数据块中存放了该目录下的所有目录项

```
00513: /*
00514:  * Structure of a directory entry
00515:  */
00516: #define EXT2_NAME_LEN 255
00517:
00518: struct ext2_dir_entry {
00519:     __le32  inode;           /* Inode number */
00520:     __le16  rec_len;         /* Directory entry length */
00521:     __le16  name_len;        /* Name length */
00522:     char    name[EXT2_NAME_LEN]; /* File name */
00523: };
```



❖ 新版的目录项结构

```
00525: /*
00526:  * The new version of the directory entry. Since EXT2 structures are
00527:  * stored in intel byte order, and the name_len field could never be
00528:  * bigger than 255 chars, it's safe to reclaim the extra byte for the
00529:  * file_type field.
00530:  */
00531: struct ext2_dir_entry_2 {
00532:     __le32  inode;           /* Inode number */
00533:     __le16  rec_len;         /* Directory entry length */
00534:     __u8    name_len;        /* Name length */
00535:     __u8    file_type;
00536:     char    name[EXT2_NAME_LEN]; /* File name */
00537: };
```



EXT2支持的文件类型

❖ EXT2在目录项中存放了文件的类型信息。文件类型可以是0~7中的任意一个整数。它们分别代表如下含义：

➤0：文件类型未知；

➤1：普通文件类型；

➤2：目录；

➤3：字符设备；

➤4：块设备；

➤5：有名管道FIFO；

➤6：套接字；

➤7：符号链接。

```
/*  
 * Ext2 directory file types. Only the low 3 bits are used. The  
 * other bits are reserved for now.  
 */  
enum {  
    EXT2_FT_UNKNOWN,  
    EXT2_FT_REG_FILE,  
    EXT2_FT_DIR,  
    EXT2_FT_CHRDEV,  
    EXT2_FT_BLKDEV,  
    EXT2_FT_FIFO,  
    EXT2_FT_SOCK,  
    EXT2_FT_SYMLINK,  
    EXT2_FT_MAX  
};
```

注意：

Type	Disk data structure	Memory data structure	Caching mode
Superblock	<code>ext2_super_block</code>	<code>ext2_sb_info</code>	Always cached
Group descriptor	<code>ext2_group_desc</code>	<code>ext2_group_desc</code>	Always cached
Block bitmap	Bit array in block	Bit array in buffer	Dynamic
inode bitmap	Bit array in block	Bit array in buffer	Dynamic
inode	<code>ext2_inode</code>	<code>ext2_inode_info</code>	Dynamic
Data block	Array of bytes	VFS buffer	Dynamic
Free inode	<code>ext2_inode</code>	None	Never
Free block	Array of bytes	None	Never



(三) 创建一个ext2文件系统

❖ 在磁盘上创建文件系统通常有两个步骤：

➤ 格式化磁盘

- Linux中：superformat或者fdformat

➤ 创建文件系统

- Ext2：mke2fs

❖ mke2fs的缺省参数

➤ 磁盘块大小：1024字节

➤ 分片：目前不支持，因此与磁盘块一样

➤ 分配inode的个数：1/8192B

➤ 永久保留的块的个数：5%



创建流程

1. 初始化超级块和组描述符
2. Optionally, 检查是否有坏块, 若有创建坏块列表
3. 对每个块组, 保留所有用来存放超级块、组描述符、inode表、2个位图的磁盘块
4. 初始化每个块组中的位图
5. 初始化每个块组中的inode表
6. 创建 */root* 目录
7. 创建 *lost+found* 目录 (供*e2fsck* 使用, 与坏块相关)
8. 为上述两个目录而更新位图信息
9. 若有坏块, 则将其在 *lost+found* 目录中组织起来



以1.44MB的软盘为例，创建ext2文件系统后

Block	Content
0	Boot block
1	Superblock
2	Block containing a single block group descriptor
3	Data block bitmap
4	inode bitmap
5-27	inode table: inodes up to 10: reserved (inode 2 is the root); inode 11: <i>lost+found</i> ; inodes 12-184: free
28	Root directory (includes <i>.</i> , <i>..</i> , and <i>lost+found</i>)
29	<i>lost+found</i> directory (includes <i>.</i> and <i>..</i>)
30-40	Reserved blocks preallocated for <i>lost+found</i> directory
41-1439	Free blocks



(四) Ext2提供的各种对象方法

❖ 超级块对象方法

```
00150: static struct super_operations ext2_sops = {  
00151:     read_inode:    ext2_read_inode,  
00152:     write_inode:   ext2_write_inode,  
00153:     put_inode:     ext2_put_inode,  
00154:     delete_inode:  ext2_delete_inode,  
00155:     put_super:     ext2_put_super,  
00156:     write_super:   ext2_write_super,  
00157:     statfs:        ext2_statfs,  
00158:     remount_fs:    ext2_remount,  
00159: };
```



❖ 索引节点对象方法

```
00052: struct inode_operations ext2_file_inode_operations = {  
00053:     truncate:    ext2_truncate,  
00054: };
```

```
00338: struct inode_operations ext2_dir_inode_operations = {  
00339:     create:      ext2_create,  
00340:     lookup:      ext2_lookup,  
00341:     link:        ext2_link,  
00342:     unlink:      ext2_unlink,  
00343:     symlink:     ext2_symlink,  
00344:     mkdir:       ext2_mkdir,  
00345:     rmdir:       ext2_rmdir,  
00346:     mknod:       ext2_mknod,  
00347:     rename:      ext2_rename,  
00348: };
```

```
00035: struct inode_operations ext2_fast_symlink_inode_operations = {  
00036:     readlink:    ext2_readlink,  
00037:     follow_link: ext2_follow_link,  
00038: };
```

❖ 文件对象方法

```
00037: /*
00038:  * We have mostly NULL's here: the current defaults are ok for
00039:  * the ext2 filesystem.
00040:  */
00041: struct file_operations ext2_file_operations = {
00042:     llseek:    generic_file_llseek,
00043:     read:      generic_file_read,
00044:     write:     generic_file_write,
00045:     ioctl:     ext2_ioctl,
00046:     mmap:      generic_file_mmap,
00047:     open:      generic_file_open,
00048:     release:   ext2_release_file,
00049:     fsync:     ext2_sync_file,
00050: };

00591: struct file_operations ext2_dir_operations = {
00592:     read:      generic_read_dir,
00593:     readdir:   ext2_readdir,
00594:     ioctl:     ext2_ioctl,
00595:     fsync:     ext2_sync_file,
00596: };
```

(五) 管理ext2的磁盘空间

- ❖ 存储在磁盘上的文件与用户所“看到”的文件有所不同：
 - 用户感觉，文件在逻辑上是连续的
 - 而在磁盘上，存储文件数据的磁盘块可能分散在磁盘各处
 - 用户感觉，文件可能比较大
 - 而在磁盘上，由于文件空洞的存在，分配给文件的磁盘空间可能小于用户感觉到的文件大小。



❖ 涉及到如下操作:

- 创建/删除一个索引节点
- 数据块的寻址
- 文件空洞
- 分配/释放一个数据块



创建/删除一个索引节点

❖ 创建一个磁盘索引节点

```
00314: struct inode * ext2_new_inode (const struct inode * dir, int mode)
```

❖ 删除一个索引节点

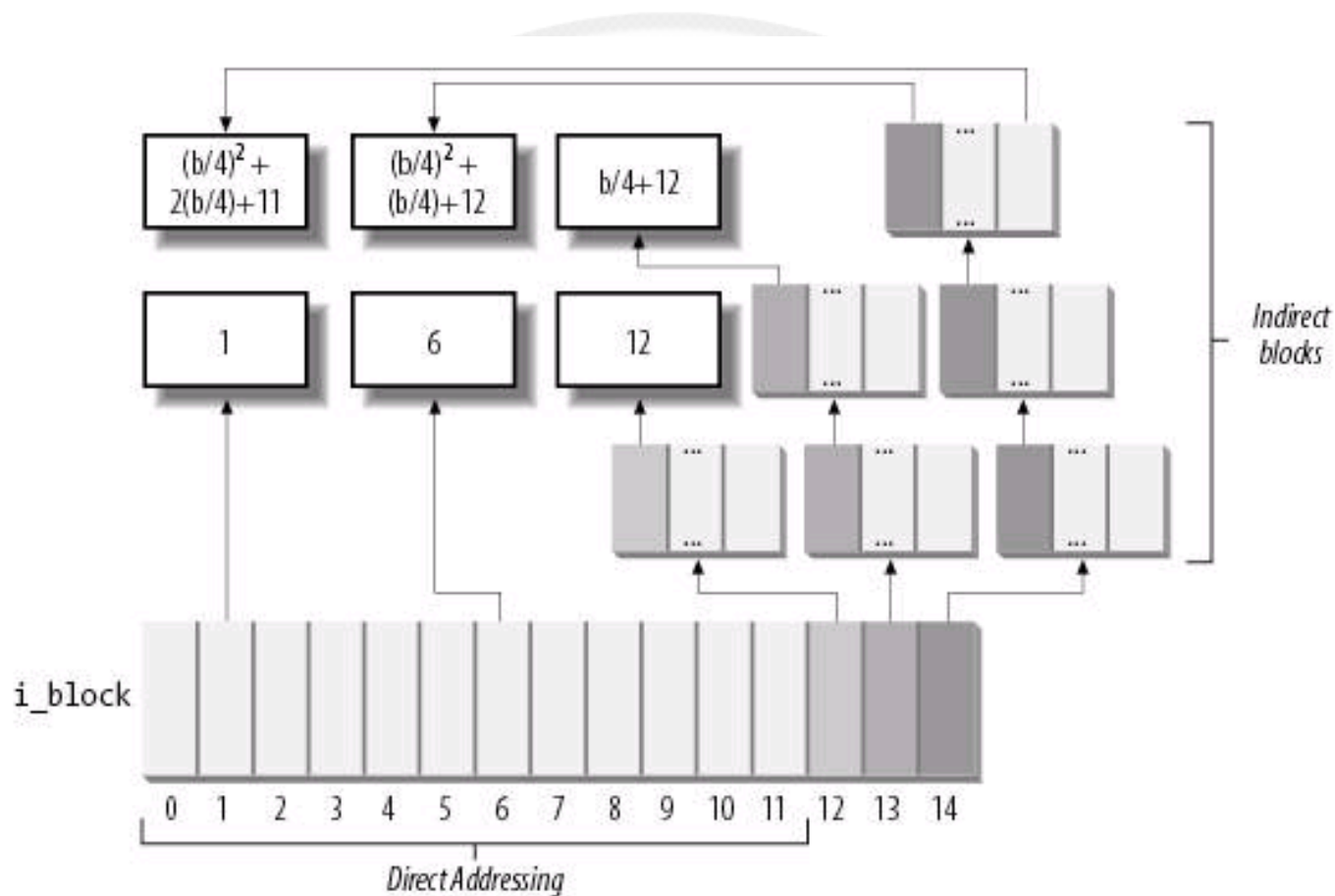
```
00133: /*
00134:  * NOTE! When we get the inode, we're the only people
00135:  * that have access to it, and as such there are no
00136:  * race conditions we have to worry about. The inode
00137:  * is not on the hash- lists, and it cannot be reached
00138:  * through the filesystem because the directory entry
00139:  * has been deleted earlier.
00140:  *
00141:  * HOWEVER: we must make sure that we get no aliases,
00142:  * which means that we have to call "clear_inode()"
00143:  * _before_ we mark the inode not in use in the inode
00144:  * bitmaps. Otherwise a newly created file might use
00145:  * the same inode number (not actually the same pointer
00146:  * though), and then we'd have two inodes sharing the
00147:  * same inode number and space on the harddisk.
00148:  */
00149: void ext2_free_inode (struct inode * inode)
```

关于数据块的寻址

- ❖ 任何一个常规文件都会包含一系列数据块
- ❖ 文件内块号 vs. 逻辑块号
 - 根据数据在文件中的偏移可以计算逻辑块号：
 - 首先计算出文件内块号 = (偏移 f - 1) / 块大小的商 + 1
 - 根据索引信息，查询到逻辑块号



混合索引示意图



文件大小限制

Block size	Direct	1-Indirect	2-Indirect	3-Indirect
1,024	12 KB	268 KB	64.26 MB	16.06 GB
2,048	24 KB	1.02 MB	513.02 MB	256.5 GB
4,096	48 KB	4.04 MB	4 GB	~ 4 TB



关于文件空洞

- ❖ A **file hole** is a portion of a regular file that contains null characters and is **not stored in any data block** on disk.
- ❖ 这是UNIX文件一直以来都有的一个特性

- ❖ 例如命令：

```
$ echo -n "X" | dd of=/tmp/hole bs=1024 seek=6
```

创建一个大小为 $1024 \times 6 + 1$ 字节的文件，这个文件有一个 $1024 \times 6 = 6144$ 个字节大小的空洞。只有最后一个字节存放了字母“X”

- ❖ 文件空洞可以节省磁盘空间
- ❖ Ext2通过数据块的动态分配来实现这一点：
当且仅当一个进程要写数据到文件中的时候才真正分配磁盘块



分配/释放一个数据块

❖ 当一个文件需要新的数据块来存放数据时

```
00494: /*
00495:  * Allocation strategy is simple: if we have to allocate something, we will
00496:  * have to go the whole way to leaf. So let's do it before attaching anything
00497:  * to tree, set linkage between the newborn blocks, write them if sync is
00498:  * required, recheck the path, free and repeat if check fails, otherwise
00499:  * set the last missing link (that will protect us from any truncate-generated
00500:  * removals - all blocks on the path are immune now) and possibly force the
00501:  * write on the parent block.
00502:  * That has a nice additional property: no special recovery from the failed
00503:  * allocations is needed - we simply release blocks and do not touch anything
00504:  * reachable from inode.
00505:  */
00506:
00507: static int ext2_get_block(struct inode *inode, long iblock, struct buffer_head *bh_result, int create)
```

❖ 当一个文件被删除或者被截断时

```
00788: void ext2_truncate (struct inode * inode)
    ↓
00753: static void ext2_free_branches(struct inode *inode, u32 *p, u32 *q, int depth)
    ↓
00250: /* Free given blocks, update quota and i_blocks field */
00251: void ext2_free_blocks (struct inode * inode, unsigned long block,
00252:                        unsigned long count)
```

Thanks!

The end.



嵌入式系统实验室

EMBEDDED SYSTEM LABORATORY
SUZHOU INSTITUTE FOR ADVANCED STUDY OF USTC