Linux 0.01 内核文件系统模块主要数据结构总结

Weball　2002.12.7

Linux0.01 内核中文件系统模块是联系内核和用户的接口，用户通过文件系统接口来对文件实现操作，同时文件系统对根据用户进程的请求来协调内存管理和内核管理模块，使整个系统能够协调工作，一般来说，大部分的系统调用都在文件系统中实现甚至本来应该在 mm 中实现的 do_exec。

首先需要了解 linux0。01 文件系统的一些主要特点，linux0。01 内核的文件系统采用的方式是 unix 的文件组织方式，而不是 windows 的组织方式，主要特点可以总结为以下几条：

1. 在整个系统中，文件的组织是一个树型结构，也就是系统在初始化的时候初始化一个 root（根节点），然后每个文件或目录都作为一个节点在 root 这颗树上，当我们要访问某个其他文件系统时，通过 mount 安装到 root 树上，而不同于 windows 的访问各个逻辑盘的方法。
2. Linux0。01 中，文件分为普通文件、目录文件、设备文件等，它把正规为文件、目录、字符设备、块设备等都当做文件来操作。
3. Linux0。01 内核中，文件的表示方式和 windows 也不相同，在 linux 中，用 inode（I 节点）来表示文件或目录，每个文件或目录通过一个唯一的 inode 号来表示，同时文件的两部分（文件名和文件的信息）分别在不同位置，文件名在它对应的目录下，文件的其他信息（如 uid、gid、修改时间、文件内容所在的块）等重要信息都在 inode 节点中。这个 windows 的 FAT 结构完全不同。
4. 文件系统中每一个文件都可以共享，在 nlinks 字段中可以联系很多的进程对此为文件操作。

下面看看 linux0。01 文件系统是一个逻辑块的组合，它可以实现在任何块设备中，它的整个布局如下：

引导块	超级块	I 节点位图	块位图	I 节点表	数据块

其中引导块（boot）放在文件系统开始，占第一个扇区，包括可执行代码，虽然一个系统只有一个引导块来引导系统，但是每个文件系统的第一块都分配给引导块。

超级块（superblock）描述整个此文件系统的状态及布局信息，它主要给出文件系统不同部分的大小，如文件多大，有多少文件可以存取，自由空块的位置等等，超级块是文件系统中非常重要的结构，在后面会详细解释。

I 节点位图是记录系统中 I 节点的状况，一般来说，一个系统能够存取的文件或目录的个数是固定的，那么这些 I 节点资源通过一个位图（就是一定的存储空间，每个位记录一个 I 节点的情况，0 表示还未用，1 表示已用），这样很多 I 节点在位图中已经记录了，同时分配或释放 I 节点实际就是对 I 节点位图进行操作和标记，例如系统有 1024 个 I 节点，那么在 I 节点位图中有定义一个 128 个字节的位图空间（128×8＝1024 位），通过这 128 个字节的位图来顺序记录后面的 I 节点的使用状态。

块位图和 I 节点位图的功能一样，是用来记录系统中块的使用情况，在文件系统中，一个块（block）的大小是 1k，不同文件系统的块大小不一样。

I 节点表是系统中所有的 I 节点表，文件的操作和访问实际是在 I 节点表中找到对应文件的 I 节点，然后根据 I 节点上的信息来访问磁盘上的数据的。在实际系统运行中，用得最

多的就是 I 节点表。

数据块是紧跟着 I 节点表的，它就是存放文件或目录的具体数据信息。

介绍了一些 linux0。01 文件系统的大体布局之后，就可以对文件系统的主要数据结构进行一些注解：

1. 在 linux 中操作文件的时候，根据文件提供的设备号（dev）来进行不同的调用和操作，其中设备号的高 8 位是主设备号，指明设备类型，这里的设备类型有 8 中，分别是：

   0－未用

   1－内存设备/dev/mem

   2－软盘/dev/fd

   3－硬盘/dev/hd

   4－终端 1/dev/ttyx

   5－终端 2/dev/tty

   6－lp 设备/dev/lp

   7－管道

   因为有 8 个设备，同时每一个设备都有一个 I 节点位图和块位图，所以在 fs.h 中定义的 I_MAP_SLOTS 和 Z_MAP_SLOTS 都是 8。

   设备号的低 8 位是次设备号，表明此设备中的一些参数。
2. Linux 中文件名长度最大是 14 个字节
3. 在文件系统中 superblock 的一个标记为 super_magic（魔数）为 0x137f，也就是在安装一个系统的时候，根据 super_magic 来判断是不是一个超级块。从而判断此文件系统的类型。
4. 在 fs.h 中有几个关于文件的参数，一个是 NR_OPEN,它是 20，它代表一个进程最多能够打开文件的个数，因为在 linux0。01 内核中，进程表关于文件系统访问的部分是在文件系统中，一个进程可以打开很多文件并且给每一个打开的文件分配一个 fd（文件描述符），NR_OPEN 就规定了一个进程最多能够打开的文件数。
5. 第二个参数是 NR_INODE,它规定了在 I 节点缓存中的 I 节点的个数，这里 I 节点缓存是在内存中为了更快的访问最近访问的 I 节点而设立的一种缓存机制，系统规定 NR_INODE 为 32，也就是说在系统的 I 节点缓存中有最多 32 个 I 节点。这里要说明的是，I 节点缓存在系统中表示为 inode_table 但是它和我们前面分析的 I 节点表不同，它是内存中的 I 节点表，而前面讲的 I 节点表是在硬盘上的 I 节点表，所以我把 inode_table 叫 I 节点缓存以备区分。
6. 第三个参数是 NR_FILE,它表明了 file_table 中的个数，也就是在整个文件表中作多打开文件的个数，这里的关系需要解释清楚，每个进程可能打开最多 20 个文件，这些文件不是直接指向 I 节点表，而是中间通过它们的 fd 指向 file_table 中的某一项，然后 file_table 才指向 I 节点表。
7. 下面的几个参数就比较简单了，NR_SUPER 说明了超级块的个数，也就是说 linux0。01 只支持 8 个超级块，也就是说 linux0。01 最多只能够 mount7 个不同的系统（因为肯定有一个 root 在启动的时候 mount 了）
8. NR_HASH 指明了 hash 表中的元素个数，系统默认是 307。NR_BUFFERS 表明系统中有多少个可以分配的缓冲，它的值在 buffer 初始化的时候按照来确定。

在对文件系统的整个全局参数了解之后，就可以具体的看看在文件系统中贯穿的几个主要的数据结构，这几个数据结构是 buffer_head, inode（包含 d_inode 和 m_inode），dir_entry, file 和 super_block，下面一一介绍。

1． Buffer_head

Buffer_head 是文件系统中用的最多的结构了，凡是对磁盘中的数据进行操作都要通过 buffer，不论是读写数据块、读写 I 节点还是查找都要经过 buffer 的缓存，所以几乎在每个函数中都会有 buffer_head 结构，因为通过一个 buffer_head 结构来表示一个 1k 的缓冲快，它的具体字段如下：

```
struct buffer_head {
    char * b_data;                  /*  指向一个 1K 数据块的指针  */
    unsigned short b_dev;           /*  此块的设备号  (如果是 0 则表明空闲) */
    unsigned short b_blocknr;       /*  表明此块的块号  */
    unsigned char b_uptodate;       /*  表明此数据是最新的和磁盘上的数据一致*/
    unsigned char b_dirt;           /*  表明此数据块是否需要写回磁盘，1 表示需要  */
    unsigned char b_count;          /*  指明当前有多少进程在使用此块  */
    unsigned char b_lock;           /*  表示当前是否有进程在操作并锁住此块，  */
    struct task_struct * b_wait;    /*  此块的等待进程，如果多个进程欲使用此块但是现
                                        在有一个进程锁住此缓冲区，就在此缓冲区等待*/
    因为每个 buffer 要么就在空闲链表中，要么就在使用链表中，所以下面就定义了
    这两个链表的指针
    struct buffer_head * b_prev;    /*  使用链表的向前指针*/
    struct buffer_head * b_next;    /*  使用链表的向后指针*/
    struct buffer_head * b_prev_free;  /*  空闲链表的向前指针*/
    struct buffer_head * b_next_free;` /*  空闲链表的向后指针*/
};
```

2． Inode 结构

I 节点是 linux 中表示一个文件或目录的主要数据结构，每个文件或目录都用一个唯一的 I 节点号来表示，所以文件系统对文件或目录的操作实际就是对 I 节点进行操作，所以 I 节点的数据结构使用非常频繁，为了加快 I 节点的访问，linux 把 inode 分为两种方式保存，一种是在硬盘中的 inode（d_inode），一种是在内存中的 inode(m_inode),内存中的 inode 除了完全包含 d_inode 中的字段之外还有一些专门的字段。

1) d_inode

```
struct d_inode {
    unsigned short i_mode;      /* I 节点的模式，如读写、执行等权限*/
                                I_mode 一共有 10 位，第一位表明 I 节点文件类型，
                                然后后 9 位依次位 I 节点所有者、文件所属组成员
                                和其他成员的访问权限，权限有读写和执行 3 种
    unsigned short i_uid;       /* I 节点的 user id*/
    unsigned long i_size;       /*  此 I 节点文件的大小*/
    unsigned long i_time;       /* I 节点创建时间*/
    unsigned char i_gid;        /* I 节点的组 id（group id）*/
```

```
    unsigned char i_nlinks;        /* 表明有多少进程链接到此 I 节点*/
    unsigned short i_zone[9];      /* 此文件数据对应在磁盘上地址*/
```

这里需要指明一点，由于在 linux0。01 内核中，数据是最终存放在磁盘中的，而磁盘存放数据的单位是块（block），一块 1k 大小，对于小于 7k 的文件，文件信息就直接可以反映在 inode 中（I 节点利用 I_zone 的前 7 个数据来存放前 7 块数据在磁盘上的地址），对于大文件，inode 通过利用间接块的方式在支持，I_zone 的第 8 个数据存放的是一次间接块的地址，一次间接块又可以存放 512 个块的数据地址，I_zone 的第 9 个数据存放的是二次间接块的地址，二次间接块又指向 512 个一次间接块，每个一次间接块又可以指向 512 个数据块，这样 linux 最大可以至此（512×512＋512＋7）×1K＝256。5M 大小的文件。

2）m_inode 结构

m_inode 是存放在内存中的 I 节点，它比 d_inode 多一些字段，具体的字段如下：

```
struct m_inode {
    unsigned short i_mode;         /*  和前面一样表示 I 节点的模式*/
    unsigned short i_uid;          /*  同前，表示 I 节点的 user id（文件所有者）*/
    unsigned long i_size;          /*  表示 I 节点文件的大小*/
    unsigned long i_mtime;         /*  表明文件上次修改时间*/
    unsigned char i_gid;           /*  I 节点的组 id，也就是文件所在的组*/
    unsigned char i_nlinks;        /*  I 节点的链接数*/
    unsigned short i_zone[9];      /*  I 节点的数据在磁盘上的对应地址*/

    以下是 d_inode 中没有的而只在内存中存在的那一部分
    struct task_struct * i_wait;   /*  I 节点的等待队列，同 buffer 中的等待队列，
                                       在管道中用到*/
    unsigned long i_atime;         /*  文件的上次访问时间*/
    unsigned long i_ctime;         /*  I 节点修改时间，注意和 I_mtime 区别*/
    unsigned short i_dev;          /*  表明此 I 节点所属设备的设备号*/
    unsigned short i_num;          /*  I 节点的 I 节点号*/
    unsigned short i_count;        /*  I 节点被打开次数，主要判断文件是否共享*/
    unsigned char i_lock;          /*  判断 I 节点是否被锁住*/
    unsigned char i_dirt;          /*  判断 I 节点是否需要写回磁盘*/
    unsigned char i_pipe;          /*  表明 I 节点是否是管道文件*/

    下面几个字段在 0。01 内核中没有用到
    unsigned char i_mount;         /*  如果有文件系统安装在此节点上，则置此位*/
    unsigned char i_seek;          /*  在 lseek 调用时置此位*/
    unsigned char i_update;
};
```

3． Dir_entry 结构

Dir_entry 结构是目录的数据结构，前面说过，文件除了文件名外信息都在 inode 中保存了，它的文件名就在 dir_entry 中，目录的数据结构中就只有一个它包含文件的文

件名和对应的 inode 号。

```
struct dir_entry
{    unsigned short inode;          /*  文件的 I 节点号*/
     char name[NAME_LEN];           /*  文件的文件名*/
};
```

4. File 结构

File 结构中就是在 file_table 中的结构，它主要是给出了文件的一些读写模式和标志，其中 file 结构中包含了一个很重要的字段就是当前读写文件的位置信息，也就是说读写文件的时候就需要根据 file 结构来找读写位置，同时 seek 调用的时候就是改变 file 结构的位置信息，具体结构如下：

```
struct file
{
 unsigned short f_mode;         /*  文件的读写模式，1 表示读，2 表示写*/
 unsigned short f_flags;        /*  对文件的操作标志，如添加、创建、修改等*/
 unsigned short f_count;        /*  此文件结构被进程引用的次数*/
 struct m_inode * f_inode;      /*  此 file 结构对应的 I 节点*/
 off_t f_pos;                   /*  当前在文件的位置标志*/
};
```

5. Super block 结构

超级块中表示了整个文件系统总体信息，包括 I 节点和数据块的个数，I 节点和数据块位图，安装信息等，具体结构如下：

```
struct super_block {
     unsigned short s_ninodes;          /*  此系统中的 I 节点个数*/
     unsigned short s_nzones;           /*  系统中的最大数据区域（包括位图）*/
     unsigned short s_imap_blocks;      /*  系统中 I 节点位图的块数*/
     unsigned short s_zmap_blocks;      /*  数据块位图所占块数*/
     unsigned short s_firstdatazone;    /*  第一数据块*/
     unsigned short s_log_zone_size;    /*  一个以 2 为底的幂来表示数据块大小*/
     unsigned long s_max_size;          /*  系统支持的最大文件大小*/
     unsigned short s_magic;            /*  此系统的 magic number 判断系统类型*/

     以下超级块字段是存储在内存中的部分而磁盘上没有
     struct buffer_head * s_imap[8];    /*  I 节点位图对应的块*/
     struct buffer_head * s_zmap[8];    /*  数据块位图对应的块*/
     unsigned short s_dev;              /*  超级块所属设备的设备号*/
     struct m_inode * s_isup;           /*  安装系统系统的根目录 I 节点*/
     struct m_inode * s_imount;         /*  文件系统在整个树上的安装节点*/
     unsigned long s_time;              /*  超级块的时间*/
     unsigned char s_rd_only;           /*  表明是否此超级块只读*/
     unsigned char s_dirt;              /*  判断此超级块是否应写回磁盘*/
};
```