代码说明文档

# 1 linux-4.10.4

## 1.1 fs

### 1.1.1 fs->f2fs

#### 1.1.1.1 fs->f2fs->node.c

**函数: bool available\_free\_memory(struct f2fs\_sb\_info \*sbi, int type)**

{

入口参数为: \*sbi, type

返回值：bool

功能：1.根据type判断是否有空闲的内存。

}

**函数: static void clear\_node\_page\_dirty(struct page \*page)**

{

入口参数为: \*page

返回值: void

功能: 1.根据page判断是否为脏页,若是则删除该页。

}

**函数：static struct page \*get\_current\_nat\_page（struct f2fs\_sb\_info \*sbi, nid\_t nid）**

{

入口参数为：\*sbi, nid。

返回值：page\*

功能：1.调用current\_nat\_addr(sbi, nid)函数，得到索引。

2.调用get\_meta\_page(sbi, index)函数，得到当前nat的page\*.

}

**函数：static struct page \*get\_next\_nat\_page（struct f2fs\_sb\_info \*sbi, nid\_t nid）**

{

入口参数为：\*sbi, nid。

返回值：page\*

功能：1.调用NM\_I(sbi)函数得到节点管理信息nm\_i。

2.得到下一个nat的page\*。

}

**函数：static struct nat\_entry \*\_\_lookup\_nat\_cache（struct f2fs\_nm\_info \*nm\_i, nid\_t n）**

{

入口参数为：\*nm\_i, n。

返回值：nat\_entry\*

功能：1.调用radix\_tree\_lookup(&nm\_i->nat\_root, n)函数，返回nat\_entry\*。

}

**函数：static unsigned int \_\_gang\_lookup\_nat\_cache（struct f2fs\_nm\_info \*nm\_i, nid\_t start, unsigned int nr, struct nat\_entry \*\*ep）**

{

入口参数为：\*nm\_i, start, nr, \*\*ep。

返回值：unsigned int

功能：1. 调用radix\_tree\_gang\_lookup(&nm\_i->nat\_root, (void \*\*)ep, start, nr)函数，返回unsigned int。

}

**函数：static void \_\_del\_from\_nat\_cache（struct f2fs\_nm\_info \*nm\_i, struct nat\_entry \*e）**

{

入口参数为：\*nm\_i, \*e。

返回值：nat\_entry\*

功能：1.调用list\_del(&e->list)。

2.调用radix\_tree\_delete(&nm\_i->nat\_root,e)函数。

3.调用kmem\_cache\_free(nat\_entry\_slab, e)函数，释放缓存。

}

**函数：static void \_\_set\_nat\_cache\_dirty（struct f2fs\_nm\_info \*nm\_i, struct nat\_entry \*ne）**

{

入口参数为：\*nm\_i, \*ne。

返回值：void

功能：1.调用NAT\_BLOCK\_OFFSET(ne->ni.nid)函数。

2.调用radix\_tree\_lookup(&nm\_i->nat\_root,set)函数。

3.调用f2fs\_kmem\_cache\_alloc(nat\_entry\_set\_slab, GFP\_NODE)函数。

4.调用INIT\_LIST\_HEAD(&head->entry\_list)。

5.调用INIT\_LIST\_HEAD(&head->set\_list)。

6.调用f2fs\_radix\_tree\_insert(&nm\_i->nat\_set\_root, set, head)函数。

7.调用list\_move\_tail(&ne->list, &head->entry\_list)。

8.调用set\_nat\_flag(ne, IS\_DIRTY, true),设置节点条目为脏。

}

**函数：static void \_\_clear\_nat\_cache\_dirty（struct f2fs\_nm\_info \*nm\_i, struct nat\_entry \*ne）**

{

入口参数为：\*nm\_i, \*ne。

返回值：void

功能：1.调用NAT\_BLOCK\_OFFSET(ne->ni.nid)函数。

2.调用radix\_tree\_lookup(&nm\_i->nat\_root,set)函数。

3.调用list\_move\_tail(&ne->list, &head->entry\_list)。

4.调用set\_nat\_flag(ne, IS\_DIRTY, false),设置节点条目不为脏。

}

**函数：static unsigned int \_\_gang\_lookup\_nat\_set（struct f2fs\_nm\_info \*nm\_i, nid\_t start, unsigned int nr, struct nat\_entry\_set \*\*ep）**

{

入口参数为：\*nm\_i, start, nr, \*\*ep。

返回值：unsigned int

功能：1.调用radix\_tree\_gang\_lookup(&nm\_i->nat\_set\_root, (void \*\*)ep, start, nr)函数在节点集合根中查找，返回unsigned int。

}

**函数：int need\_denty\_mark(struct f2fs\_sb\_info \*sbi, nid\_t nid)**

{

入口参数为：\*sbi, nid。

返回值：int

功能：1.调用NM\_I(sbi)函数得到nm\_i。

2.调用down\_read(&nm\_i->nat\_tree\_lock).

3.调用\_\_lookup\_nat\_cache（nm\_i,nid）得到nat\_entry指针。

4.调用函数进行判断。

5.调用up\_read（&nm\_i->nat\_tree\_lock）

}

**函数：bool is\_checkpointed\_node（struct f2fs\_sb\_info \*sbi, nid\_t nid）**

{

入口参数为：\*sbi, nid

返回值：bool

功能：1.调用NM\_I(sbi)函数得到nm\_i。

2.调用down\_read(&nm\_i->nat\_tree\_lock)

3.调用\_\_lookup\_nat\_cache（nm\_i,nid）得到nat\_entry指针。

4.调用函数进行判断。

5.调用up\_read（&nm\_i->nat\_tree\_lock）

}

**函数：bool need\_inode\_block\_update（struct f2fs\_sb\_info \*sbi, nid\_t ino）**

{

入口参数为：\*sbi, ino

返回值：bool

功能：1.调用NM\_I(sbi)函数得到nm\_i。

2.调用down\_read(&nm\_i->nat\_tree\_lock)

3.调用\_\_lookup\_nat\_cache（nm\_i,nid）得到nat\_entry指针。

4.调用函数进行判断。

5.调用up\_read（&nm\_i->nat\_tree\_lock）

}

**函数：static struct nat\_entry \*grab\_nat\_entry（struct f2fs\_nm\_info \*nm\_i, nid\_t nid）**

{

入口参数为：\*nm\_i,nid。

返回值：nat\_entry\*

功能：1.调用f2fs\_kmem\_cache\_alloc(nat\_entry\_slab, GFP\_NOFS)函数在内核内存中分配一个新的nat\_entry。

2.调用f2fs\_radix\_tree\_insert(&nm\_i->nat\_root, nid, new)函数将新的nat\_entry插入到radix\_tree中。

3.调用nat\_set\_nid(new, nid)函数为新的nat\_entry设置nid。

4.调用nat\_reset\_flag(new)函数为新的nat\_entry重置flag。

5.调用list\_add\_tail(&new->list, &nm\_i->nat\_entries)函数将新的nat\_entry添加到nm\_i中。

}

**函数：static void cache\_nat\_entry(struct f2fs\_sb\_info \*sbi, nid\_t nid, struct f2fs\_nat\_entry \*ne)**

{

入口参数为：\*sbi, nid, \*ne

返回值：void

功能：1.调用NM\_I(sbi)函数得到nm\_i。

2.调用\_\_lookup\_nat\_cache（nm\_i,nid）得到nat\_entry指针。

3.调用函数进行判断。

}

**函数：static void set\_node\_addr(struct f2fs\_sb\_info \*sbi, struct node\_info \*ni, block\_t new\_blkaddr, bool fsync\_done)**

{

入口参数为：\*sbi, \*ni, new\_blkaddr, fsync\_done

返回值：void

功能：1.调用NM\_I(sbi)函数得到nm\_i。

2.调用down\_write(&nm\_i->nat\_tree\_lock)。

2.调用\_\_lookup\_nat\_cache（nm\_i,ni->nid）得到nat\_entry指针。

3.调用函数进行判断，当节点id被分配，之前的nat\_entryt可以被保留在nat cache中，因此，用新的信息初始。

4.进行完整性检查。

5.当节点移除时增加版本号。

6.调用nat\_set\_blkaddr(e, new\_blkaddr)函数改变地址。

7.调用up\_write(&nm\_i->nat\_tree\_lock)函数

}

**函数：int try\_to\_free\_nats(struct f2fs\_sb\_info \*sbi, int nr\_shrink)**

{

入口参数为：\*sbi, nr\_shrink

返回值：void

功能：1.调用NM\_I(sbi)函数得到nm\_i。

3.调用函数进行判断，如果nat\_entries链表不为空，则将nat\_entries链表的第一个entry取出。

4.调用\_\_del\_from\_nat\_cache(nm\_i, ne)函数，将entry从缓存中删除。

5.调用up\_write(&nm\_i->nat\_tree\_lock)函数

}

**函数：void try\_to\_free\_nats(struct f2fs\_sb\_info \*sbi, int nr\_shrink)**

{

入口参数为：\*sbi, nr\_shrink

返回值：void

功能：1.调用NM\_I(sbi)函数得到nm\_i。

3.调用函数进行判断，如果nat\_entries链表不为空，则将nat\_entries链表的第一个entry取出。

4.调用\_\_del\_from\_nat\_cache(nm\_i, ne)函数，将entry从缓存中删除。

5.调用up\_write(&nm\_i->nat\_tree\_lock)函数

}

**函数：void get\_node\_info(struct f2fs\_sb\_info \*sbi, struct node\_info \*ni)**

{

入口参数为：\*sbi, \*ni

返回值：void

功能：1.调用NM\_I(sbi)函数得到nm\_i。

3.调用CURSEG\_I(sbi, CURSEG\_HOT\_DATA)函数得到当前SEG信息。

4.检查nat cache。

5.检查当前segment的摘要。

6.从nat page填node\_info。

7.cache nat entry。

}

**函数：static void ra\_node\_page(struct page \*parent, int start, int n)**

{

入口参数为：\*parent, start, n

返回值：void

功能：1.调用F2FS\_P\_SB(parent)函数得到sbi。

2.调用blk\_start\_plug(&plug)函数。

3.然后调用get\_nid(parent, i, false)函数和ra\_node\_page(sbi,nid)函数进行预读。

4.调用blk\_finish\_plug(&plug)。

}

**函数：pgoff get\_next\_page\_offset(struct dnode\_of\_data \*dn, pgoff\_t pgofs)**

{

入口参数为：\*dn, pgofs

返回值：void

功能：1.返回下一页的偏移。

}

**函数：static int get\_node\_path(struct inode \*inode, long block, int offset[4], unsigned int noffset[4])**

{

入口参数为：\*inode, block, offset[4], noffset[4]

返回值：int

功能：1.返回节点的深度。

}

**函数：int get\_dnode\_of\_data(struct dnode\_of\_data \*dn, pgoff\_t index, int mode)**

{

入口参数为：\*dn, index, mode

返回值：int

功能：1.得到目录节点数据。

}

**函数：static void truncate\_node(struct dnode\_of\_data \*dn)**

{

入口参数为：\*dn

返回值：void

功能：1.缩短节点。

}

**函数：static int truncate\_dnode(struct dnode\_of\_data \*dn)**

{

入口参数为：\*dn

返回值：int

功能：1.缩短目录节点。

}

未完待续。。。

#### 1.1.1.2 fs->f2fs->inode.c

**函数：void f2fs\_mark\_inode\_dirty\_sync(struct inode \*inode, bool sync)**

{

入口参数为：\*inode, sync

返回值：void

功能：1.将索引节点标记为脏。

}

**函数：void f2fs\_set\_inode\_flag(struct inode \*inode)**

{

入口参数为：\*inode

返回值：void

功能：1.设置索引节点的flag。

}

**函数：void \_\_get\_inode\_rdev(struct inode \*inode, struct f2fs\_inode \*ri)**

{

入口参数为：\*inode, \*ri

返回值：void

功能：1.得到索引节点的译码设备。

}

**函数：void \_\_get\_inode\_rdev(struct inode \*inode, struct f2fs\_inode \*ri)**

{

入口参数为：\*inode, \*ri

返回值：void

功能：1.得到索引节点的译码设备。

}

**函数：bool \_\_written\_first\_block(struct f2fs\_inode \*ri)**

{

入口参数为：\*inode, \*ri

返回值：bool

功能：1.若inode page的直接数据块地址不是新或者空地址，则返回true。

}

**函数：static void \_\_set\_inode\_rdev(struct inode \*inode, struct f2fs\_inode \*ri)**

{

入口参数为：\*inode, \*ri

返回值：void

功能：1.根据条件设置inode page的直接数据块1和2的地址

}

**函数：static void \_\_recover\_inline\_status(struct inode \*inode, struct page \*ipage)**

{

入口参数为：\*inode, \*ipage

返回值：void

功能：1.将内联数据中的页覆盖到f2fs中的页内。

}

**函数：static int do\_read\_inode(struct inode \*inode)**

{

入口参数为：\*inode

返回值：int

功能：1.将f2fs中的索引节点信息读取到上层的索引节点中。

}

**函数：struct inode \*f2fs\_iget(struct super\_block \*sb, unsigned long ino)**

{

入口参数为：\*sb, ino

返回值：inode \*

功能：1.根据sb和ino信息来设置inode的成员变量值并返回。

}

**函数：struct inode \*f2fs\_iget\_retry(struct super\_block \*sb, unsigned long ino)**

{

入口参数为：\*sb, ino

返回值：inode \*

功能：1.当系统拥塞时，等待一段时间尝试获得inode。

}

**函数：int update\_inode(struct inode \*inode, struct page \*node\_page)**

{

入口参数为：\*inode, \*node\_page

返回值：int

功能：1.用inode的信息更新f2fs\_inode的信息。

}

**函数：int update\_inode\_page(struct inode \*inode)**

{

入口参数为：\*inode,

返回值：int

功能：1.调用update\_inode(inode, node\_page)函数，用inode的信息更新f2fs\_inode的信息。

}

**函数：int f2fs\_write\_inode(struct inode \*inode, struct writeback\_control \*wbc)**

{

入口参数为：\*inode, \*wbc

返回值：int

功能：1.调用update\_inode(inode, node\_page)函数。

2.为了避免产生脏的node pages，调用f2fs\_balance\_fs(abi, true)函数。

}

**函数：void f2fs\_evict\_inode(struct inode \*inode)**

{

入口参数为：\*inode

返回值：int

功能：1.进行一系列操作后将inode清除。

}

**函数：void handle\_failed\_inode(struct inode \*inode)**

{

入口参数为：\*inode

返回值：void

功能：1.调用clear\_nlink(inode)函数。

2.调用update\_inode\_page(inode)函数。

3.调用unlock\_new\_inode(inode)函数。

4.调用get\_node\_info(sbi, inode->i\_ino, &ni)函数。

5.调用f2fs\_unlock\_op(sbi)函数。

6.调用input(inode)函数。

}

#### 1.1.1.3 fs->f2fs->segment.c

**函数：static unsigned long \_\_reverse\_ulong(unsigned char \*str)**

{

入口参数为：\*str

返回值：unsigned long

功能：1.颠倒无符号长整形数。

}

**函数：static incline unsigned long \_\_reverse\_ffs(unsigned long word)**

{

入口参数为：word

返回值：unsigned long

功能：1.根据参数word，返回一个无符号长整形数。

}

**函数：static unsigned long \_\_find\_rev\_next\_bit(const unsigned long \*addr, unsigned long size, unsigned long offset)**

{

入口参数为：\*addr, size, offset

返回值：unsigned long

功能：1.位操作函数，具体功能待用时分析。

}

**函数：static unsigned long \_\_find\_rev\_next\_zero\_bit(const unsigned long \*addr, unsigned long size, unsigned long offset)**

{

入口参数为：\*addr, size, offset

返回值：unsigned long

功能：1.位操作函数，具体功能待用时分析。

}

**函数：void register\_inmem\_page(struct inode \*inode, struct page \*page)**

{

入口参数为：\*inode, \*page

返回值：void

功能：1.向内存中登记一个page。

}

**函数：static int \_\_revoke\_inmem\_pages(struct inode \*inode, struct list\_head \*head, bool drop, bool recover)**

{

入口参数为：\*inode, \*head, drop, recover

返回值：int

功能：1.撤回内存中一个page,具体功能待用时分析。

}

**函数：void drop\_inmem\_pages(struct inode \*inode)**

{

入口参数为：\*inode

返回值：void

功能：1.调用\_\_revoke\_inmem\_pages()函数。

}

**函数：static int \_\_commit\_inmem\_pages(struct inode \*inode, struct list\_head \*revoke\_list)**

{

入口参数为：\*inode, \*revoke\_list

返回值：int

功能：1.提交一个内存page。。

}

**函数：int commit\_inmem\_pages(struct inode \*inode)**

{

入口参数为：\*inode

返回值：int

功能：1.调用\_\_commit\_inmem\_pages()函数,若失败则调用\_\_revoke\_inmem\_page（）函数撤回。

}

**函数：void f2fs\_balance\_fs(struct f2fs\_sb\_info \*sbi, bool need)**

{

入口参数为：\*sbi, need

返回值：void

功能：1.该函数的功能是平衡脏的节点和目录项页。

2.控制gc。

}

**函数：void f2fs\_balance\_fs\_bg(struct f2fs\_sb\_info \*sbi)**

{

入口参数为：\*sbi

返回值：void

功能：1.该函数的功能是平衡脏的节点和目录项页。

}

**函数：static int \_\_submit\_flush\_wait(struct block\_device \*bdev)**

{

入口参数为：\*bedv

返回值：int

功能：1.调用f2fs\_bio\_alloc(0)函数分配bio。

2.调用submit\_bio\_wait(bio)函数,提交bio等待。

3.调用bio\_put(bio)将bio放进等待队列中。

}

**函数：static int submit\_flush\_wait(struct f2fs\_sb\_info \*sbi)**

{

入口参数为：\*sbi

返回值：int

功能：1.调用\_\_submit\_flush\_wait()函数。

}