struct address\_space {

struct xarray i\_pages;

};

struct msdos\_inode\_info {

loff\_t mmu\_private;

};

文件系统的struct address\_space\_operations中实现了回调write\_begin和write\_end，

## write\_begin

write\_begin主要分配struct address\_space的对应的page，最新内核的page由struct xarray管理。并分配page对应的struct buffer\_head。

### write\_begin会分配物理扇区和buffer\_head吗

答案是会的。

fat\_write\_begin->

cont\_write\_begin->

block\_write\_begin(mapping,pos,len,flags,pagep,get\_block)

{

pgoff\_t index = pos >> PAGE\_SHIFT;

int status;

page = grab\_cache\_page\_write\_begin(mapping,index,flags);

status = \_\_block\_write\_begin(page,pos,len,get\_block);

\*pagep = page;

return status;

}

\_\_block\_write\_begin(page,pos,len,get\_block)

->\_\_block\_write\_begin\_int(page,pos,len,get\_block,NULL)

{

}

### lseek后，文件的pos超过物理分配的扇区write\_begin会如何处理呢

在fat32中，struct msdos\_inode\_info的mmc\_private成员保存着实际分配的物理扇区大小。

fat\_write\_begin->

cont\_write\_begin->

cont\_expand\_zero

在fat\_write\_begin中会将当前文件的pos和当前文件实际分配的物理扇区大小（文件大小，而不是簇大小）传入到cont\_expand\_zero中，cont\_expand\_zero会做如下事情:

#define PAGE\_MASK (~((1<<PAGE\_SHIFT)-1))

/\*

如果当前pos大于文件的实际大小，需要调用cont\_expand\_zero将当前文件实际大小和pos之间的数据填0，其原理是从文件实际大小开始，到pos，先处理非blocksize和PAGE\_SIZE对齐，然后调用write\_begin和write\_end清零。

pos大于文件的实际大小场景只会出现在lseek上。

\*/

cont\_expand\_zero(file,mapping,pos,bytes)

{

pgoff\_t index,curidx;

unsigned zerofrom,offset,len;

unsigned len;

index = pos>>PAGE\_SHIFT;

offset = pos & ~PAGE\_MASK;

struct page \*page;

while(index > (curidx = (curpos =\*bytes)>>PAGE\_SHIFT)) {

/\*

获取page区域，也就是0-11bit

\*/

zerofrom = curpos &~PAGE\_MASK;

/\*

如果zerofrom不是512字节对齐，则需要向上对齐到512字节

\*/

if(zerofrom &(blocksize-1)) {

\*bytes |= (blocksize-1);

(\*byte)++;

}

/\*

获取zerofrom离整个PAGE\_SIZE还有多大

\*/

len = PAGE\_SIZE – zerofrom;

err = pagecache\_write\_begin(file,mapping,curpos,len,0,&page,&fsdata);

zero\_user(page,zerofrom,len);

err = pagecache\_write\_end(file,mapping,curpos,len,len,page,fsdata);

}

if(index == curidx) {

zerofrom = curpos&~PAGE\_MASK；

if(offset <= zerofrom)

goto out;

if(zerofrom &(block\_size-1)) {

\*bytes |= (blocksize-1);

(\*bytes)++;

}

len = offset – zerofrom;

err = pagecache\_write\_begin();

zero\_user(page,zerofrom,len);

err = pagecache\_write\_begin(file,mapping,curpos,len,len,page,fsdata);

err =0;

}

out:

return err;

}

## write\_end