wb\_workfn

struct wb\_writeback\_work {

unsigned int for\_kupdate:1;

unsigned long \*older\_than\_this;

};

struct bdi\_writeback {

struct delayed\_work dwork;

struct list\_head work\_list;

struct list\_head b\_io;

struct list\_head b\_dirty;

struct list\_head b\_more\_io;

struct list\_head b\_dirty\_time;

};

struct inode {

struct list\_head i\_io\_list;

unsigned long dirtied\_when;

struct super\_block \*sb;

struct list\_head i\_lru;

};

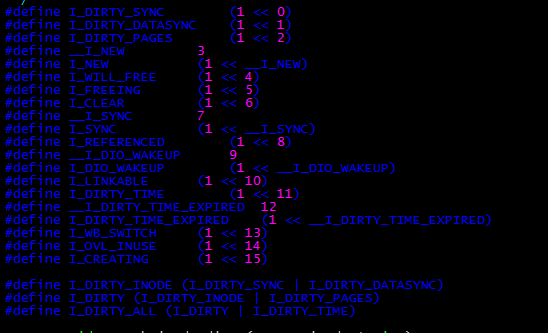
struct writeback\_control {

unsigned for\_sync:1;/\*sync(2) WB\_SYNC\_ALL writeback\*/

}

mm/backing-dev.c

在wb\_init中初始化为struct bdi\_writeback的dwork



I\_DIRTY\_SYNC和I\_DIRTY\_DATASYNC区分开来，I\_DIRTY\_SYNC是说inode dirty了，但是是i\_atime之类的dirty;I\_DIRTY\_DATASYNC是说inode其他地方dirty了；I\_DIRTY\_PAGES是说inode的有效数据dirty了；I\_DIRTY\_TIME是说inode的时间dirty了。

## wb\_workfn

wb\_workfn(work)

{

struct bdi\_writeback \*wb = container\_of(to\_delayed\_work(work),struct bdi\_writeback,dwork);

long pages\_written;

if(!current\_is\_workqueue\_rescuer()) {

do {

pages\_written = wb\_do\_writeback(wb);

}while(!list\_empty(&wb->work\_list));

}

}

wb\_do\_writeback(wb)

{

struct wb\_writeback\_work \*work;

long wrote = 0;

while((work = get\_next\_work\_item(wb))!=NULL)

{

wrote += wb\_writeback(wb,work);

}

}

get\_next\_work\_item(wb)

{

struct wb\_writeback\_work \*work = NULL;

if(!list\_empty(&wb->work\_list)) {

work = list\_entry(wb->work\_list.next,struct wb\_writeback\_work,list);

list\_del\_init(&work->list);

}

return work;

}

wb\_writeback(wb,work)

{

long progress;

for(;;)

{

if(work->nr\_pages<=0)

break;

if(list\_empty(&wb->b\_io))

queue\_io(wb,work);

if(work->sb)

progress = writeback\_sb\_inodes(work->sb,wb,work);

}

}

queue\_io(wb,work)

{

int move;

/\*

将b\_more\_io放到b\_io链表上

\*/

list\_splice\_init(&wb->b\_more\_io,&wb->io);

move = move\_expired\_inodes(&wb->b\_dirty,&wb->b\_io,0,work);

}

move\_expired\_inodes(&wb->b\_dirty,&wb->b\_io,0,work)

{

struct list\_head \*delaying\_queue = &wb->b\_dirty ;

int flags =0;

struct list\_head \*dispatch\_queue = &wb->b\_io;

unsigned long \*older\_than\_this = NULL;

unsigned long expire\_time;

struct inode \*inode;

LIST\_HEAD(tmp);

int moved =0;

if((flags&EXPIRE\_DIRTY\_ATIME)==0)

older\_than\_this = work->older\_than\_this;

else if(!work->for\_sync) {

expire\_time = jiffies – (dirtytime\_expire\_interval \*HZ);

older\_than\_this = &expire\_time;

}

while(!list\_empty(delaying\_queue)) {

/\*

获取tail的inode

\*/

inode = wb\_inode(delaying\_queue->prev);

/\*

delaying\_queue最新的会插在表头，最老的会在表尾，如果最老的都在older\_than\_this的后面，说明当前列表都在older\_than\_this后面，则没有满足要求的，需要退出。

\*/

if(older\_than\_this && inode\_dirtied\_after(inode,\*older\_than\_this)) {

break;

}

list\_move(&inode->i\_io\_list,&tmp);

moved++;

}

return moved;

}

writeback\_sb\_inodes(work->sb,wb,work)

{

struct writeback\_control wbc = {

};

long write\_chunk;

long wrote =0;

unsigned long start\_time = jiffies;

while(!list\_empty(&wb->b\_io)) {

struct inode \*inode = wb\_inode(wb->b\_io.prev);

if(inode->i\_state &(I\_NEW | I\_FREEING | I\_WILL\_FREE))

{

}

if(inode->i\_state &I\_SYNC) {

inode\_sleep\_on\_writeback(inode);

continue;

}

/\*

I\_SYNC说明正在writeback阶段

\*/

inode->i\_state |= I\_SYNC;

/\*

如果未定义CONFIG\_CGROUP\_WRITEBACK，wbc\_attach\_and\_unlock\_inode只是执行spin\_unlock(&inode->i\_lock)

\*/

wbc\_attach\_and\_unlock\_inode(&wbc,inode)

write\_chunk = writeback\_chunk\_size(wb,work);

wbc.nr\_to\_write = write\_chunk;

wbc.pages\_skipped = 0;

\_\_writeback\_single\_inode(inode,&wbc);

/\*

wbc\_detach\_inode如果CONFIG\_CGROUP\_WRITEBAK未定义，则不执行任何操作

\*/

wbc\_detach\_inode(&wbc);

tmp\_wb = inode\_to\_wb\_and\_lock\_list(inode);

if(!(inode->i\_state&I\_DIRTY\_ALL))

wrote++;

requeue\_inode(inode,tmp\_wb,&wbc);

inode\_sync\_complete(inode);

if(wrote) {

if(time\_is\_before\_jiffies(start\_time+HZ/10UL))

break;

if(work->nr\_pages<=0)

break;

}

}

return wrote;

}

requeue\_inode(inode,tmp\_wb,&wbc)

{

struct bdi\_writeback \*wb = tmp\_wb;

struct writeback\_control \*wbc = wbc;

}

## \_\_writeback\_single\_inode

<https://lwn.net/Articles/631469/>

\_\_writeback\_single\_inode(inode,wbc)

{

int ret;

struct address\_space \*mapping = inode->i\_mapping;

ret = do\_writepages(mapping,wbc);

/\*

先写数据，才能写元数据。

\*/

if(wbc->sync\_mode == WB\_SYNC\_ALL &&!wbc->for\_sync) {

int err = filemap\_fdatawait(mapping);

if(ret ==0)

ret =err;

}

/\*

I\_DIRTY = I\_DIRTY\_SYNC | I\_DIRTY\_DATASYNC；代表的是inode的dirty

\*/

dirty = inode->i\_state & I\_DIRTY;

/\*

\*/

if(inode->i\_state &I\_DIRTY\_TIME) {

if((dirty&(I\_DIRTY\_SYNC | I\_DIRTY\_DATASYNC)) ||

wbc->sync\_mode ==WB\_SYNC\_ALL) {

dirty |= I\_DIRTY\_TIME |I\_DIRTY\_TIME\_EXPIRED;

}

}else

inode->i\_state &=~I\_DIRTY\_TIME\_EXPIRED；

inode->i\_state &= ~dirty;

if(mapping\_tagged(mapping,PAGECACHE\_TAG\_DIRTY))

{

inode->i\_state |=I\_DIRTY\_PAGES;

}

if(dirty & I\_DIRTY\_TIME)

mark\_inode\_dirty\_sync(inode);

if(dirty &~I\_DIRTY\_PAGES) {

}

}

\_\_mark\_inode\_dirty()

{

}