## open

## write\_begin write\_end

#define EXT4\_INODE\_JOURNAL\_DATA\_MODE 0x01

#define EXT4\_INODE\_ORDERED\_DATA\_MODE 0x02

#define EXT4\_INODE\_WRITEBACK\_DATA\_MODE 0x04

typedef struct jbd2\_journal\_handle handle\_t;

typedef struct transaction\_s transaction\_t;

struct transaction\_s {

journal\_t \*t\_journal;

enum {

T\_RUNNING,

T\_LOCKED,

T\_SWITCH,

T\_FLUSH,

T\_COMMIT,

T\_COMMIT\_DFLUSH,

T\_COMMIT\_JFLUSH,

T\_COMMIT\_CALLBACK,

T\_FINISHED

}t\_state;

struct journal\_head \*t\_buffers;

unsigned long t\_requested;

struct list\_head t\_inode\_list;

};

struct jbd2\_journal\_handle {

union {

transaction\_t \*h\_transaction;

journal\_t \*h\_journal;

};

unsigned long h\_start\_jiffies;

};

struct journal\_s journal\_t;

struct journal\_s {

transaction\_t \*j\_running\_transaction;

transaction\_t \*j\_committing\_transaction;

/\*

@j\_wbuf:array of bhs for jbd2\_journal\_commit\_transaction

\*/

struct buffer\_head \*\*j\_wbuf;

};

ext4\_write\_begin(file,mapping,pos,len,flags,pagep,fsdata)

{

handle\_t \*handle;

int needed\_blocks;

handle = ext4\_journal\_start(inode,EXT4\_HT\_WRITE\_PAGE,needed\_blocks);

if(!ret &&ext4\_should\_journal\_data(inode)) {

ret = ext4\_walk\_page\_buffers(handle,page\_buffers(page),from,to,NULL,

do\_journal\_get\_write\_access);

}  
}

ext4\_write\_end(file,mapping,pos,len,copied,page,fsdata)

{

if(i\_size\_changed )

ext4\_mark\_inode\_dirty(handle,inode);

ext4\_journal\_stop(handle);

}

ext4\_journal\_stop(handle)->

\_\_ext4\_journal\_stop(\_\_function\_\_,\_\_line\_\_,handle)

{

jbd2\_journal\_stop(handle);

}  
jbd2\_journal\_stop(handle)

{

transaction\_t \*transaction = handle->h\_transaction;

journal\_t \*journal;

pid = current->pid

if(handle->h\_sync&&journal->j\_last\_sync\_writer !=pid)

{

if(handle->h\_sync || atomic\_read(&transaction->t\_outstanding\_credits) > journal->j\_max\_transaction\_buffers)

{

jbd2\_log\_start\_commit(journal,transaction->t\_tid);

}

}   
}

jbd2\_log\_start\_commit(journal,transaction->t\_tid)

{

if(journal->j\_running\_transaction&&journal->j\_running\_transaction->t\_tid==target)

{

wake\_up(&journal->j\_wait\_commit);

return 1;

}

}

ext4\_mark\_inode\_dirty(handle,inode)

{

struct ext4\_iloc iloc;

ext4\_reserve\_inode\_write(handle,inode,&iloc);

}

ext4\_reserve\_inode\_write(handle,inode,&iloc)

{

/\*

获取inode的位置

\*/

ext4\_get\_inode\_loc(inode,iloc)  
 ext4\_journal\_get\_write\_access(handle,iloc->bh);

}

ext4\_journal\_get\_write\_access(handle,iloc->bh)->

\_\_ext4\_journal\_get\_write\_access(\_\_function\_\_,\_\_line\_\_,handle,bh)->

jbd2\_journal\_get\_write\_access(handle,bh)

{

struct journal\_head \*jh;

if(jbd2\_write\_access\_granted(handle,bh,false))

return 0;

/\*

分配struct journal\_head并将struct buffer\_head 加入到里面

\*/

jh = jbd2\_journal\_add\_journal\_head(bh);

do\_get\_write\_access(handle,jh,0);

}

jbd2\_write\_access\_granted(handle,bh,false)

{

struct journal\_head \*jh;

if(buffer\_dirty(bh))

return false;

}

jbd2\_journal\_add\_journal\_head(bh)

{

struct journal\_head \*jh;

struct journal\_head \*new\_jh = NULL;

if(!buffer\_jbd(bh))

new\_jh = journal\_alloc\_journal\_head();

jh = new\_jh;

new\_jh = NULL;

set\_buffer\_jbd(bh);

bh->b\_private = jh;

jh->b\_bh = bh;

get\_bh(bh);

return bh->b\_private;

}

do\_get\_write\_access(handle,jh,0)

{

transaction\_t \*transaction = handle->h\_transaction;

journal\_t \*journal;

journal = transaction->t\_journal;

repeat:

bh = jh2bh(jh);

if(buffer\_dirty(bh)) {

}

if(!jh->transaction) {

\_\_jbd2\_journal\_file\_buffer(jh,transaction,BJ\_Reserved);

}

}

\_\_jbd2\_journal\_file\_buffer(jh,transaction,BJ\_Reserved)

{

struct journal\_head \*\*list = NULL;

int jlist = BJ\_Reserved;

jh->b\_transaction = transaction;

switch(jlist) {

list = &transaction->t\_reserved\_list;

break;

}

\_\_blist\_add\_buffers(list,jh);

jh->b\_jlist = jlist;

}

## kjournald2

kjournald2(void \*arg)

{

loop:

if(journal->j\_commit\_sequence != journal->j\_commit\_request) {

del\_timer\_sync(&journal->j\_commit\_timer);

jbd2\_journal\_commit\_transaction(journal);

}

DEFINE\_WAIT(wait);

int should\_sleep;

prepare\_to\_wait(&journal->j\_wait\_commit,&wait,TASK\_INTERRUPTIBLE);

if(journal->j\_commit\_sequence != journal->j\_commit\_request)

should\_sleep =0;

transaction = journal->j\_running\_transaction;

if(transaction &&time\_after\_eq(jiffies,transaction->t\_expires))

should\_sleep = 0;

if(journal->j\_flags &JBD2\_UMMOUNT)

should\_sleep = 0;

if(should\_sleep) {

schedule();

}

finish\_wait(&journal->j\_wait\_commit,&wait);

goto loop;

}

jbd2\_journal\_commit\_transaction(journal)

{

transaction\_t \*commit\_transaction;

struct buffer\_head \*descriptor;

char \*tagp = NULL;

struct buffer\_head \*\*wbuf = journal->j\_wbuf;

int bufs;

LIST\_HEAD(log\_bufs);

commit\_transaction = journal->j\_running\_transaction;

commit\_transaction->t\_state = T\_LOCKED;

commit\_transaction->t\_state = T\_SWITCH;

while(commit\_transaction->t\_reserved\_list) {

jh = commit\_transaction->t\_reserved\_list;

jbd2\_journal\_refile\_buffer(journal,jh);

}

commit\_transaction->t\_state = T\_FLUSH;

journal->j\_commit\_transaction = commit\_transaction;

journal->j\_running\_transaction = NULL;

if(journal->j\_flags &JBD2\_FLUSHED)

{

jbd2\_journal\_update\_sb\_log\_tail();

}

journal\_submit\_data\_buffers(journal,commit\_transaction);

descriptor = NULL;

while(commit\_transactions->t\_buffers) {

jh = commit\_transactions->t\_buffers;

if(!descriptor) {

descriptor = jbd2\_journal\_get\_descriptor\_buffer(commit\_transaction,

JBD2\_DESCRIPTOR\_BLOCK);

tagp = &descriptor->b\_data[sizeof(journal\_header\_t)];

space\_left = descriptor->b\_size – sizeof(journal\_header\_t);

first\_tag = 1;

set\_buffer\_jwrite(descriptor);

set\_buffer\_dirty(descriptor);

wbuf[bufs++] = descriptor;

jbd2\_file\_log\_bh(&log\_bufs,descriptor);

}

jbd2\_journal\_next\_log\_block(journal,&blocknr);

flags = jbd2\_journal\_write\_metadata\_buffer(commit\_transaction,jh,&wbuf[bufs],blocknr);

}

}

journal\_submit\_data\_buffers(journal,commit\_transaction)

{

struct jbd2\_inode \*jinode;

struct address\_space \*mapping;

list\_for\_each\_entry(jinode,&commit\_transaction->t\_inode\_list,i\_list) {

loff\_t dirty\_start = jinode->i\_dirty\_start;

loff\_t dirty\_end = jinode->i\_dirty\_end;

mapping = jinode->i\_vfs\_inode->i\_mapping;

jinode->i\_flags |= JI\_COMMIT\_RUNNING;

journal\_submit\_inode\_data\_buffers(mapping,dirty\_start,dirty\_end);

}

}

jbd2\_journal\_write\_metadata\_buffer(commit\_transaction,jh,&wbuf[bufs],blocknr)

{

journal\_t \*journal = transaction->t\_journal;

struct buffer\_head \*new\_bh;

}

jbd2\_journal\_get\_descriptor\_buffer(commit\_transaction,JBD2\_DESCRIPTOR\_BLOCK)

{

journal\_t \*journal = transaction->t\_journal;

unsigned long long blocknr;

struct buffer\_head \*bh;

journal\_header\_t \*header;

jbd2\_journal\_next\_log\_block(journal,&blocknr);

bh = \_\_getblk(journal->j\_dev,blocknr,journal->j\_blocksize);

memset(bh->b\_data,0,journal->j\_blocksize);

header = (journal\_header\_t \*)bh->b\_data;

header->h\_magic = cpu\_to\_be32(JBD2\_MAGIC\_NUMBER);

header->h\_blocktype = cpu\_to\_be32(JBD2\_DESCRIPTOR\_BLOCK);

header->h\_sequence = cpu\_to\_be32(transaction->t\_tid);

set\_buffer\_uptodate(bh);

return bh;

}

jbd2\_file\_log\_bh(&log\_bufs,descriptor)

{

list\_add\_tail(&bh->b\_assoc\_buffers,head);

}

## writepages

/\*

delayed allocation stuff

\*/

struct ext4\_io\_submit {

struct writeback\_control \*io\_wbc;

ext4\_io\_end\_t \*io\_end;

struct bio \*io\_bio;

};

struct mpage\_da\_data {

struct ext4\_io\_submit io\_submit;

};

ext4\_writepages(mapping,wbc)

{

struct mpage\_da\_data mpd;

mpd.first\_page = wbc->range\_start >> PAGE\_SHIFT;

mpd.last\_page = wbc->range\_end >> PAGE\_SHIFT;

while(!done &&mpd.first\_page <= mpd.last\_page) {

handle= ext4\_journal\_start\_with\_reserve(inode,EXT4\_HT\_WRITE\_PAGE,needed\_blocks,rsv\_blocks);

}

}

ext4\_journal\_start\_with\_reserve(inode,EXT4\_HT\_WRITE\_PAGE,needed\_blocks,rsv\_blocks)

{

}