LCOV - code coverage report

Current view: directory - fs/ext4 - ext4_jbd2.h (source / functions)		Found	Hit
Test: kernel_2_6_31_ext4_round_3.info	Lines:	57	36
Date: 2009-10-24	Functions:	0	0

```
                :  /*
                :   * ext4_jbd2.h
                :   *
                :   * Written by Stephen C. Tweedie <sct@redhat.com>, 1999
                :   *
                :   * Copyright 1998--1999 Red Hat corp --- All Rights Reserved
                :   *
                :   * This file is part of the Linux kernel and is made available under
                :   * the terms of the GNU General Public License, version 2, or at your
                :   * option, any later version, incorporated herein by reference.
                :   *
                :   * Ext4-specific journaling extensions.
                :   */
                :
                :  #ifndef _EXT4_JBD2_H
                :  #define _EXT4_JBD2_H
                :
                :  #include <linux/fs.h>
                :  #include <linux/jbd2.h>
                :  #include "ext4.h"
                :
                :  #define EXT4_JOURNAL(inode)     (EXT4_SB((inode)->i_sb)->s_journal)
                :
                :  /* Define the number of blocks we need to account to a transaction to
                :   * modify one block of data.
                :   *
                :   * We may have to touch one inode, one bitmap buffer, up to three
                :   * indirection blocks, the group and superblock summaries, and the data
                :   * block to complete the transaction.
                :   *
                :   * For extents-enabled fs we may have to allocate and modify up to
                :   * 5 levels of tree + root which are stored in the inode. */
                :
                :  #define EXT4_SINGLEDATA_TRANS_BLOCKS(sb)                                \
                :          (EXT4_HAS_INCOMPAT_FEATURE(sb, EXT4_FEATURE_INCOMPAT_EXTENTS)    \
                :           ? 27U : 8U)
                :
                :  /* Extended attribute operations touch at most two data buffers,
                :   * two bitmap buffers, and two group summaries, in addition to the inode
                :   * and the superblock, which are already accounted for. */
                :
                :  #define EXT4_XATTR_TRANS_BLOCKS         6U
                :
                :  /* Define the minimum size for a transaction which modifies data.  This
                :   * needs to take into account the fact that we may end up modifying two
                :   * quota files too (one for the group, one for the user quota).  The
                :   * superblock only gets updated once, of course, so don't bother
                :   * counting that again for the quota updates. */
                :
                :  #define EXT4_DATA_TRANS_BLOCKS(sb)      (EXT4_SINGLEDATA_TRANS_BLOCKS(sb) + \
                :                                           EXT4_XATTR_TRANS_BLOCKS - 2 + \
                :                                           2*EXT4_QUOTA_TRANS_BLOCKS(sb))
```

```
53                :
54                : /*
55                :  * Define the number of metadata blocks we need to account to modify data.
56                :  *
57                :  * This include super block, inode block, quota blocks and xattr blocks
58                :  */
59                : #define EXT4_META_TRANS_BLOCKS(sb)      (EXT4_XATTR_TRANS_BLOCKS + \
60                :                                         2*EXT4_QUOTA_TRANS_BLOCKS(sb))
61                :
62                : /* Delete operations potentially hit one directory's namespace plus an
63                :  * entire inode, plus arbitrary amounts of bitmap/indirection data.  Be
64                :  * generous.  We can grow the delete transaction later if necessary. */
65                :
66                : #define EXT4_DELETE_TRANS_BLOCKS(sb)    (2 * EXT4_DATA_TRANS_BLOCKS(sb) + 64)
67                :
68                : /* Define an arbitrary limit for the amount of data we will anticipate
69                :  * writing to any given transaction.  For unbounded transactions such as
70                :  * write(2) and truncate(2) we can write more than this, but we always
71                :  * start off at the maximum transaction size and grow the transaction
72                :  * optimistically as we go. */
73                :
74                : #define EXT4_MAX_TRANS_DATA             64U
75                :
76                : /* We break up a large truncate or write transaction once the handle's
77                :  * buffer credits gets this low, we need either to extend the
78                :  * transaction or to start a new one.  Reserve enough space here for
79                :  * inode, bitmap, superblock, group and indirection updates for at least
80                :  * one block, plus two quota updates.  Quota allocations are not
81                :  * needed. */
82                :
83                : #define EXT4_RESERVE_TRANS_BLOCKS       12U
84                :
85                : #define EXT4_INDEX_EXTRA_TRANS_BLOCKS   8
86                :
87                : #ifdef CONFIG_QUOTA
88                : /* Amount of blocks needed for quota update - we know that the structure was
89                :  * allocated so we need to update only inode+data */
90                : #define EXT4_QUOTA_TRANS_BLOCKS(sb) (test_opt(sb, QUOTA) ? 2 : 0)
91                : /* Amount of blocks needed for quota insert/delete - we do some block writes
92                :  * but inode, sb and group updates are done only once */
93                : #define EXT4_QUOTA_INIT_BLOCKS(sb) (test_opt(sb, QUOTA) ? (DQUOT_INIT_ALLOC*\
94                :                 (EXT4_SINGLEDATA_TRANS_BLOCKS(sb)-3)+3+DQUOT_INIT_REWRITE) : 0)
95                : #define EXT4_QUOTA_DEL_BLOCKS(sb) (test_opt(sb, QUOTA) ? (DQUOT_DEL_ALLOC*\
96                :                 (EXT4_SINGLEDATA_TRANS_BLOCKS(sb)-3)+3+DQUOT_DEL_REWRITE) : 0)
97                : #else
98                : #define EXT4_QUOTA_TRANS_BLOCKS(sb) 0
99                : #define EXT4_QUOTA_INIT_BLOCKS(sb) 0
100               : #define EXT4_QUOTA_DEL_BLOCKS(sb) 0
101               : #endif
102               :
103               : int
104               : ext4_mark_iloc_dirty(handle_t *handle,
105               :                      struct inode *inode,
106               :                      struct ext4_iloc *iloc);
107               :
108               : /*
109               :  * On success, We end up with an outstanding reference count against
110               :  * iloc->bh.  This _must_ be cleaned up later.
111               :  */
112               :
113               : int ext4_reserve_inode_write(handle_t *handle, struct inode *inode,
114               :                              struct ext4_iloc *iloc);
115               :
116               : int ext4_mark_inode_dirty(handle_t *handle, struct inode *inode);
117               :
```

```
   118                :  /*
   119                :   * Wrapper functions with which ext4 calls into JBD.  The intent here is
   120                :   * to allow these to be turned into appropriate stubs so ext4 can control
   121                :   * ext2 filesystems, so ext2+ext4 systems only nee one fs.  This work hasn't
   122                :   * been done yet.
   123                :   */
   124                :
   125                : void ext4_journal_abort_handle(const char *caller, const char *err_fn,
   126                :                 struct buffer_head *bh, handle_t *handle, int err);
   127                :
   128                : int __ext4_journal_get_undo_access(const char *where, handle_t *handle,
   129                :                                    struct buffer_head *bh);
   130                :
   131                : int __ext4_journal_get_write_access(const char *where, handle_t *handle,
   132                :                                    struct buffer_head *bh);
   133                :
   134                : /* When called with an invalid handle, this will still do a put on the BH */
   135                : int __ext4_journal_forget(const char *where, handle_t *handle,
   136                :                                    struct buffer_head *bh);
   137                :
   138                : /* When called with an invalid handle, this will still do a put on the BH */
   139                : int __ext4_journal_revoke(const char *where, handle_t *handle,
   140                :                                    ext4_fsblk_t blocknr, struct buffer_head *bh);
   141                :
   142                : int __ext4_journal_get_create_access(const char *where,
   143                :                                    handle_t *handle, struct buffer_head *bh);
   144                :
   145                : int __ext4_handle_dirty_metadata(const char *where, handle_t *handle,
   146                :                                     struct inode *inode, struct buffer_head *bh);
   147                :
   148                : #define ext4_journal_get_undo_access(handle, bh) \
   149                :         __ext4_journal_get_undo_access(__func__, (handle), (bh))
   150                : #define ext4_journal_get_write_access(handle, bh) \
   151                :         __ext4_journal_get_write_access(__func__, (handle), (bh))
   152                : #define ext4_journal_revoke(handle, blocknr, bh) \
   153                :         __ext4_journal_revoke(__func__, (handle), (blocknr), (bh))
   154                : #define ext4_journal_get_create_access(handle, bh) \
   155                :         __ext4_journal_get_create_access(__func__, (handle), (bh))
   156                : #define ext4_journal_forget(handle, bh) \
   157                :         __ext4_journal_forget(__func__, (handle), (bh))
   158                : #define ext4_handle_dirty_metadata(handle, inode, bh) \
   159                :         __ext4_handle_dirty_metadata(__func__, (handle), (inode), (bh))
   160                :
   161                : handle_t *ext4_journal_start_sb(struct super_block *sb, int nblocks);
   162                : int __ext4_journal_stop(const char *where, handle_t *handle);
   163                :
   164                : #define EXT4_NOJOURNAL_HANDLE   ((handle_t *) 0x1)
   165                :
   166                : static inline int ext4_handle_valid(handle_t *handle)
   167                : {
   168             -1 :         if (handle == EXT4_NOJOURNAL_HANDLE)
   169        3510150 :                 return 0;
   170             -1 :         return 1;
   171                : }
   172                :
   173                : static inline void ext4_handle_sync(handle_t *handle)
   174                : {
   175              0 :         if (ext4_handle_valid(handle))
   176              0 :                 handle->h_sync = 1;
   177                : }
   178                :
   179                : static inline void ext4_handle_release_buffer(handle_t *handle,
   180                :                                               struct buffer_head *bh)
   181                : {
   182              0 :         if (ext4_handle_valid(handle))
```

```
   183                0 :                 jbd2_journal_release_buffer(handle, bh);
   184                  : }
   185                  :
   186                  : static inline int ext4_handle_is_aborted(handle_t *handle)
   187                  : {
   188                0 :         if (ext4_handle_valid(handle))
   189                0 :                 return is_handle_aborted(handle);
   190                0 :         return 0;
   191                  : }
   192                  :
   193                  : static inline int ext4_handle_has_enough_credits(handle_t *handle, int needed)
   194                  : {
   195          1664364 :         if (ext4_handle_valid(handle) && handle->h_buffer_credits < needed)
   196                0 :                 return 0;
   197          1664364 :         return 1;
   198                  : }
   199                  :
   200                  : static inline void ext4_journal_release_buffer(handle_t *handle,
   201                  :                                                struct buffer_head *bh)
   202                  : {
   203                  :         if (ext4_handle_valid(handle))
   204                  :                 jbd2_journal_release_buffer(handle, bh);
   205                  : }
   206                  :
   207                  : static inline handle_t *ext4_journal_start(struct inode *inode, int nblocks)
   208                  : {
   209      -2084236281 :         return ext4_journal_start_sb(inode->i_sb, nblocks);
   210                  : }
   211                  :
   212                  : #define ext4_journal_stop(handle) \
   213                  :         __ext4_journal_stop(__func__, (handle))
   214                  :
   215                  : static inline handle_t *ext4_journal_current_handle(void)
   216                  : {
   217      -1638211072 :         return journal_current_handle();
   218                  : }
   219                  :
   220                  : static inline int ext4_journal_extend(handle_t *handle, int nblocks)
   221                  : {
   222                0 :         if (ext4_handle_valid(handle))
   223                0 :                 return jbd2_journal_extend(handle, nblocks);
   224                0 :         return 0;
   225                  : }
   226                  :
   227                  : static inline int ext4_journal_restart(handle_t *handle, int nblocks)
   228                  : {
   229                0 :         if (ext4_handle_valid(handle))
   230                0 :                 return jbd2_journal_restart(handle, nblocks);
   231                0 :         return 0;
   232                  : }
   233                  :
   234                  : static inline int ext4_journal_blocks_per_page(struct inode *inode)
   235                  : {
   236        419851556 :         if (EXT4_JOURNAL(inode) != NULL)
   237        209925778 :                 return jbd2_journal_blocks_per_page(inode);
   238                0 :         return 0;
   239                  : }
   240                  :
   241                  : static inline int ext4_journal_force_commit(journal_t *journal)
   242                  : {
   243            13486 :         if (journal)
   244            13486 :                 return jbd2_journal_force_commit(journal);
   245                0 :         return 0;
   246                  : }
   247                  :
```

```
       248                :static inline int ext4_jbd2_file_inode(handle_t *handle, struct inode *inode)
       249                :{
       250        218010749:        if (ext4_handle_valid(handle))
       251        218010749:                return jbd2_journal_file_inode(handle, &EXT4_I(inode)->jinode);
       252                0:        return 0;
       253                :}
       254                :
       255                :/* super.c */
       256                :int ext4_force_commit(struct super_block *sb);
       257                :
       258                :static inline int ext4_should_journal_data(struct inode *inode)
       259                :{
       260     1215800200:        if (EXT4_JOURNAL(inode) == NULL)
       261           1233:                return 0;
       262      607898867:        if (!S_ISREG(inode->i_mode))
       263        1316753:                return 1;
       264     1213164228:        if (test_opt(inode->i_sb, DATA_FLAGS) == EXT4_MOUNT_JOURNAL_DATA)
       265       17116759:                return 1;
       266      589465355:        if (EXT4_I(inode)->i_flags & EXT4_JOURNAL_DATA_FL)
       267                0:                return 1;
       268      589465355:        return 0;
       269                :}
       270                :
       271                :static inline int ext4_should_order_data(struct inode *inode)
       272                :{
       273       75190228:        if (EXT4_JOURNAL(inode) == NULL)
       274           2025:                return 0;
       275       37593089:        if (!S_ISREG(inode->i_mode))
       276         655410:                return 0;
       277       36937679:        if (EXT4_I(inode)->i_flags & EXT4_JOURNAL_DATA_FL)
       278                0:                return 0;
       279       73875358:        if (test_opt(inode->i_sb, DATA_FLAGS) == EXT4_MOUNT_ORDERED_DATA)
       280       28471851:                return 1;
       281        8465828:        return 0;
       282                :}
       283                :
       284                :static inline int ext4_should_writeback_data(struct inode *inode)
       285                :{
       286        6640492:        if (!S_ISREG(inode->i_mode))
       287                0:                return 0;
       288       13280984:        if (EXT4_JOURNAL(inode) == NULL)
       289            105:                return 1;
       290        6640387:        if (EXT4_I(inode)->i_flags & EXT4_JOURNAL_DATA_FL)
       291                0:                return 0;
       292       13280774:        if (test_opt(inode->i_sb, DATA_FLAGS) == EXT4_MOUNT_WRITEBACK_DATA)
       293        1214285:                return 1;
       294        5426102:        return 0;
       295                :}
       296                :
       297                :#endif  /* _EXT4_JBD2_H */
```