LCOV - code coverage report

			Found	Hit	Cov
Current view:	directory - fs/ext4 - balloc.c (source / functions)				
Test:	kernel_2_6_31_ext4_round_3.info	**Lines:**	263	174	6
Date:	2009-10-24	**Functions:**	15	14	9

```
                :  1 /*
                :  2  *  linux/fs/ext4/balloc.c
                :  3  *
                :  4  * Copyright (C) 1992, 1993, 1994, 1995
                :  5  * Remy Card (card@masi.ibp.fr)
                :  6  * Laboratoire MASI - Institut Blaise Pascal
                :  7  * Universite Pierre et Marie Curie (Paris VI)
                :  8  *
                :  9  *  Enhanced block allocation by Stephen Tweedie (sct@redhat.com), 1993
                : 10  *  Big-endian to little-endian byte-swapping/bitmaps by
                : 11  *        David S. Miller (davem@caip.rutgers.edu), 1995
                : 12  */
                : 13
                : 14 #include <linux/time.h>
                : 15 #include <linux/capability.h>
                : 16 #include <linux/fs.h>
                : 17 #include <linux/jbd2.h>
                : 18 #include <linux/quotaops.h>
                : 19 #include <linux/buffer_head.h>
                : 20 #include "ext4.h"
                : 21 #include "ext4_jbd2.h"
                : 22 #include "mballoc.h"
                : 23
                : 24 /*
                : 25  * balloc.c contains the blocks allocation and deallocation routines
                : 26  */
                : 27
                : 28 /*
                : 29  * Calculate the block group number and offset, given a block number
                : 30  */
                : 31 void ext4_get_group_no_and_offset(struct super_block *sb, ext4_fsblk_t blocknr,
                : 32                 ext4_group_t *blockgrpp, ext4_grpblk_t *offsetp)
      840646490 : 33 {
      840646490 : 34         struct ext4_super_block *es = EXT4_SB(sb)->s_es;
                : 35         ext4_grpblk_t offset;
                : 36
      840646490 : 37         blocknr = blocknr - le32_to_cpu(es->s_first_data_block);
      840646490 : 38         offset = do_div(blocknr, EXT4_BLOCKS_PER_GROUP(sb));
      840646490 : 39         if (offsetp)
      457680391 : 40                 *offsetp = offset;
      840646490 : 41         if (blockgrpp)
      842392191 : 42                 *blockgrpp = blocknr;
                : 43
      840646490 : 44 }
                : 45
                : 46 static int ext4_block_in_group(struct super_block *sb, ext4_fsblk_t block,
                : 47                         ext4_group_t block_group)
                : 48 {
                : 49         ext4_group_t actual_group;
      374798056 : 50         ext4_get_group_no_and_offset(sb, block, &actual_group, NULL);
      374798056 : 51         if (actual_group == block_group)
            0 : 52                 return 1;
      374798056 : 53         return 0;
                : 54 }
                : 55
                : 56 static int ext4_group_used_meta_blocks(struct super_block *sb,
                : 57                                         ext4_group_t block_group,
```

```
    58                :                                                struct ext4_group_desc *gdp)
    59       1550884  : {
    60                :         ext4_fsblk_t tmp;
    61       1550884  :         struct ext4_sb_info *sbi = EXT4_SB(sb);
    62                :         /* block bitmap, inode bitmap, and inode table blocks */
    63       1550884  :         int used_blocks = sbi->s_itb_per_group + 2;
    64                :
    65       1550884  :         if (EXT4_HAS_INCOMPAT_FEATURE(sb, EXT4_FEATURE_INCOMPAT_FLEX_BG)) {
    66       3101768  :                 if (!ext4_block_in_group(sb, ext4_block_bitmap(sb, gdp),
    67                :                                          block_group))
    68       1550884  :                         used_blocks--;
    69                :
    70       3101768  :                 if (!ext4_block_in_group(sb, ext4_inode_bitmap(sb, gdp),
    71                :                                          block_group))
    72       1550884  :                         used_blocks--;
    73                :
    74       1550884  :                 tmp = ext4_inode_table(sb, gdp);
    75     345174728  :                 for (; tmp < ext4_inode_table(sb, gdp) +
    76     342072960  :                                 sbi->s_itb_per_group; tmp++) {
    77     342072960  :                         if (!ext4_block_in_group(sb, tmp, block_group))
    78     342072960  :                                 used_blocks -= 1;
    79                :                 }
    80                :         }
    81       1550884  :         return used_blocks;
    82                : }
    83                :
    84                : /* Initializes an uninitialized block bitmap if given, and returns the
    85                :  * number of blocks free in the group. */
    86                : unsigned ext4_init_block_bitmap(struct super_block *sb, struct buffer_head *bh,
    87                :                 ext4_group_t block_group, struct ext4_group_desc *gdp)
    88       1550884  : {
    89                :         int bit, bit_max;
    90       1550884  :         ext4_group_t ngroups = ext4_get_groups_count(sb);
    91                :         unsigned free_blocks, group_blocks;
    92       1550884  :         struct ext4_sb_info *sbi = EXT4_SB(sb);
    93                :
    94       1550884  :         if (bh) {
    95        112720  :                 J_ASSERT_BH(bh, buffer_locked(bh));
    96                :
    97                :                 /* If checksum is bad mark all blocks used to prevent allocation
    98                :                  * essentially implementing a per-group read-only flag. */
    99        112720  :                 if (!ext4_group_desc_csum_verify(sbi, block_group, gdp)) {
   100             0  :                         ext4_error(sb, __func__,
   101                :                                    "Checksum bad for group %u", block_group);
   102             0  :                         ext4_free_blks_set(sb, gdp, 0);
   103             0  :                         ext4_free_inodes_set(sb, gdp, 0);
   104             0  :                         ext4_itable_unused_set(sb, gdp, 0);
   105             0  :                         memset(bh->b_data, 0xff, sb->s_blocksize);
   106             0  :                         return 0;
   107                :                 }
   108        225440  :                 memset(bh->b_data, 0, sb->s_blocksize);
   109                :         }
   110                :
   111                :         /* Check for superblock and gdt backups in this group */
   112       1550884  :         bit_max = ext4_bg_has_super(sb, block_group);
   113                :
   114       1550884  :         if (!EXT4_HAS_INCOMPAT_FEATURE(sb, EXT4_FEATURE_INCOMPAT_META_BG) ||
   115                :             block_group < le32_to_cpu(sbi->s_es->s_first_meta_bg) *
   116                :                           sbi->s_desc_per_block) {
   117       1550884  :                 if (bit_max) {
   118             0  :                         bit_max += ext4_bg_num_gdb(sb, block_group);
   119             0  :                         bit_max +=
   120                :                                 le16_to_cpu(sbi->s_es->s_reserved_gdt_blocks);
   121                :                 }
   122                :         } else { /* For META_BG_BLOCK_GROUPS */
   123             0  :                 bit_max += ext4_bg_num_gdb(sb, block_group);
   124                :         }
   125                :
   126       1550884  :         if (block_group == ngroups - 1) {
   127                :                 /*
   128                :                  * Even though mke2fs always initialize first and last group
```

```
    129                 :                  * if some other tool enabled the EXT4_BG_BLOCK_UNINIT we need
    130                 :                  * to make sure we calculate the right free blocks
    131                 :                  */
    132               0 :                 group_blocks = ext4_blocks_count(sbi->s_es) -
    133                 :                         le32_to_cpu(sbi->s_es->s_first_data_block) -
    134                 :                         (EXT4_BLOCKS_PER_GROUP(sb) * (ngroups - 1));
    135                 :         } else {
    136         1550884 :                 group_blocks = EXT4_BLOCKS_PER_GROUP(sb);
    137                 :         }
    138                 :
    139         1550884 :         free_blocks = group_blocks - bit_max;
    140                 :
    141         1550884 :         if (bh) {
    142                 :                 ext4_fsblk_t start, tmp;
    143          112720 :                 int flex_bg = 0;
    144                 :
    145          112720 :                 for (bit = 0; bit < bit_max; bit++)
    146               0 :                         ext4_set_bit(bit, bh->b_data);
    147                 :
    148          112720 :                 start = ext4_group_first_block_no(sb, block_group);
    149                 :
    150          112720 :                 if (EXT4_HAS_INCOMPAT_FEATURE(sb,
    151                 :                                               EXT4_FEATURE_INCOMPAT_FLEX_BG))
    152          112720 :                         flex_bg = 1;
    153                 :
    154                 :                 /* Set bits for block and inode bitmaps, and inode table */
    155          112720 :                 tmp = ext4_block_bitmap(sb, gdp);
    156          225440 :                 if (!flex_bg || ext4_block_in_group(sb, tmp, block_group))
    157               0 :                         ext4_set_bit(tmp - start, bh->b_data);
    158                 :
    159          112720 :                 tmp = ext4_inode_bitmap(sb, gdp);
    160          225440 :                 if (!flex_bg || ext4_block_in_group(sb, tmp, block_group))
    161               0 :                         ext4_set_bit(tmp - start, bh->b_data);
    162                 :
    163          112720 :                 tmp = ext4_inode_table(sb, gdp);
    164        29623328 :                 for (; tmp < ext4_inode_table(sb, gdp) +
    165        29397888 :                                      sbi->s_itb_per_group; tmp++) {
    166        58795776 :                         if (!flex_bg ||
    167                 :                                 ext4_block_in_group(sb, tmp, block_group))
    168               0 :                                 ext4_set_bit(tmp - start, bh->b_data);
    169                 :                 }
    170                 :                 /*
    171                 :                  * Also if the number of blocks within the group is
    172                 :                  * less than the blocksize * 8 ( which is the size
    173                 :                  * of bitmap ), set rest of the block bitmap to 1
    174                 :                  */
    175          112720 :                 mark_bitmap_end(group_blocks, sb->s_blocksize * 8, bh->b_data);
    176                 :         }
    177         1550884 :         return free_blocks - ext4_group_used_meta_blocks(sb, block_group, gdp);
    178                 : }
    179                 :
    180                 :
    181                 : /*
    182                 :  * The free blocks are managed by bitmaps.  A file system contains several
    183                 :  * blocks groups.  Each group contains 1 bitmap block for blocks, 1 bitmap
    184                 :  * block for inodes, N blocks for the inode table and data blocks.
    185                 :  *
    186                 :  * The file system contains group descriptors which are located after the
    187                 :  * super block.  Each descriptor contains the number of the bitmap block and
    188                 :  * the free blocks count in the block.  The descriptors are loaded in memory
    189                 :  * when a file system is mounted (see ext4_fill_super).
    190                 :  */
    191                 :
    192                 :
    193                 : #define in_range(b, first, len) ((b) >= (first) && (b) <= (first) + (len) - 1)
    194                 :
    195                 : /**
    196                 :  * ext4_get_group_desc() -- load group descriptor from disk
    197                 :  * @sb:                 super block
    198                 :  * @block_group:        given block group
    199                 :  * @bh:                 pointer to the buffer head to store the block
```

```
   200                :  *                        group descriptor
   201                :  */
   202                : struct ext4_group_desc * ext4_get_group_desc(struct super_block *sb,
   203                :                                              ext4_group_t block_group,
   204                :                                              struct buffer_head **bh)
   205             -1 : {
   206                :         unsigned int group_desc;
   207                :         unsigned int offset;
   208             -1 :         ext4_group_t ngroups = ext4_get_groups_count(sb);
   209                :         struct ext4_group_desc *desc;
   210             -1 :         struct ext4_sb_info *sbi = EXT4_SB(sb);
   211                :
   212             -1 :         if (block_group >= ngroups) {
   213              0 :                 ext4_error(sb, "ext4_get_group_desc",
   214                :                            "block_group >= groups_count - "
   215                :                            "block_group = %u, groups_count = %u",
   216                :                            block_group, ngroups);
   217                :
   218              0 :                 return NULL;
   219                :         }
   220                :
   221             -1 :         group_desc = block_group >> EXT4_DESC_PER_BLOCK_BITS(sb);
   222             -1 :         offset = block_group & (EXT4_DESC_PER_BLOCK(sb) - 1);
   223             -1 :         if (!sbi->s_group_desc[group_desc]) {
   224              0 :                 ext4_error(sb, "ext4_get_group_desc",
   225                :                            "Group descriptor not loaded - "
   226                :                            "block_group = %u, group_desc = %u, desc = %u",
   227                :                            block_group, group_desc, offset);
   228              0 :                 return NULL;
   229                :         }
   230                :
   231             -1 :         desc = (struct ext4_group_desc *)(
   232                :                 (__u8 *)sbi->s_group_desc[group_desc]->b_data +
   233                :                 offset * EXT4_DESC_SIZE(sb));
   234             -1 :         if (bh)
   235      235915938 :                 *bh = sbi->s_group_desc[group_desc];
   236             -1 :         return desc;
   237                : }
   238                :
   239                : static int ext4_valid_block_bitmap(struct super_block *sb,
   240                :                                    struct ext4_group_desc *desc,
   241                :                                    unsigned int block_group,
   242                :                                    struct buffer_head *bh)
   243           4488 : {
   244                :         ext4_grpblk_t offset;
   245                :         ext4_grpblk_t next_zero_bit;
   246                :         ext4_fsblk_t bitmap_blk;
   247                :         ext4_fsblk_t group_first_block;
   248                :
   249           4488 :         if (EXT4_HAS_INCOMPAT_FEATURE(sb, EXT4_FEATURE_INCOMPAT_FLEX_BG)) {
   250                :                 /* with FLEX_BG, the inode/block bitmaps and itable
   251                :                  * blocks may not be in the group at all
   252                :                  * so the bitmap validation will be skipped for those groups
   253                :                  * or it has to also read the block group where the bitmaps
   254                :                  * are located to verify they are set.
   255                :                  */
   256           4487 :                 return 1;
   257                :         }
   258              1 :         group_first_block = ext4_group_first_block_no(sb, block_group);
   259                :
   260                :         /* check whether block bitmap block number is set */
   261              1 :         bitmap_blk = ext4_block_bitmap(sb, desc);
   262              1 :         offset = bitmap_blk - group_first_block;
   263              2 :         if (!ext4_test_bit(offset, bh->b_data))
   264                :                 /* bad block bitmap */
   265              0 :                 goto err_out;
   266                :
   267                :         /* check whether the inode bitmap block number is set */
   268              1 :         bitmap_blk = ext4_inode_bitmap(sb, desc);
   269              1 :         offset = bitmap_blk - group_first_block;
   270              2 :         if (!ext4_test_bit(offset, bh->b_data))
```

```
   271                :                 /* bad block bitmap */
   272              0 :                 goto err_out;
   273                :
   274                :         /* check whether the inode table block number is set */
   275              1 :         bitmap_blk = ext4_inode_table(sb, desc);
   276              1 :         offset = bitmap_blk - group_first_block;
   277              2 :         next_zero_bit = ext4_find_next_zero_bit(bh->b_data,
   278                :                                 offset + EXT4_SB(sb)->s_itb_per_group,
   279                :                                 offset);
   280              2 :         if (next_zero_bit >= offset + EXT4_SB(sb)->s_itb_per_group)
   281                :                 /* good bitmap for inode tables */
   282              1 :                 return 1;
   283                :
   284              0 : err_out:
   285              0 :         ext4_error(sb, __func__,
   286                :                            "Invalid block bitmap - "
   287                :                            "block_group = %d, block = %llu",
   288                :                            block_group, bitmap_blk);
   289              0 :         return 0;
   290                : }
   291                : /**
   292                :  * ext4_read_block_bitmap()
   293                :  * @sb:                 super block
   294                :  * @block_group:        given block group
   295                :  *
   296                :  * Read the bitmap for a given block_group,and validate the
   297                :  * bits for block/inode/inode tables are set in the bitmaps
   298                :  *
   299                :  * Return buffer_head on success or NULL in case of failure.
   300                :  */
   301                : struct buffer_head *
   302                : ext4_read_block_bitmap(struct super_block *sb, ext4_group_t block_group)
   303      231211575 : {
   304                :         struct ext4_group_desc *desc;
   305      231211575 :         struct buffer_head *bh = NULL;
   306                :         ext4_fsblk_t bitmap_blk;
   307                :
   308      231211575 :         desc = ext4_get_group_desc(sb, block_group, NULL);
   309      232844958 :         if (!desc)
   310              0 :                 return NULL;
   311      232844958 :         bitmap_blk = ext4_block_bitmap(sb, desc);
   312      232971226 :         bh = sb_getblk(sb, bitmap_blk);
   313      232971226 :         if (unlikely(!bh)) {
   314              0 :                 ext4_error(sb, __func__,
   315                :                            "Cannot read block bitmap - "
   316                :                            "block_group = %u, block_bitmap = %llu",
   317                :                            block_group, bitmap_blk);
   318              0 :                 return NULL;
   319                :         }
   320                :
   321      231711558 :         if (bitmap_uptodate(bh))
   322      233862602 :                 return bh;
   323                :
   324                :         lock_buffer(bh);
   325          83292 :         if (bitmap_uptodate(bh)) {
   326             18 :                 unlock_buffer(bh);
   327             18 :                 return bh;
   328                :         }
   329                :         ext4_lock_group(sb, block_group);
   330          83274 :         if (desc->bg_flags & cpu_to_le16(EXT4_BG_BLOCK_UNINIT)) {
   331          78786 :                 ext4_init_block_bitmap(sb, bh, block_group, desc);
   332                :                 set_bitmap_uptodate(bh);
   333                :                 set_buffer_uptodate(bh);
   334                :                 ext4_unlock_group(sb, block_group);
   335          78786 :                 unlock_buffer(bh);
   336          78786 :                 return bh;
   337                :         }
   338                :         ext4_unlock_group(sb, block_group);
   339           4488 :         if (buffer_uptodate(bh)) {
   340                :                 /*
   341                :                  * if not uninit if bh is uptodate,
```

```
   342                :                  * bitmap is also uptodate
   343                :                  */
   344                :                 set_bitmap_uptodate(bh);
   345              0 :                 unlock_buffer(bh);
   346              0 :                 return bh;
   347                :         }
   348                :         /*
   349                :          * submit the buffer_head for read. We can
   350                :          * safely mark the bitmap as uptodate now.
   351                :          * We do it here so the bitmap uptodate bit
   352                :          * get set with buffer lock held.
   353                :          */
   354                :         set_bitmap_uptodate(bh);
   355           4488 :         if (bh_submit_read(bh) < 0) {
   356                :                 put_bh(bh);
   357              0 :                 ext4_error(sb, __func__,
   358                :                            "Cannot read block bitmap - "
   359                :                            "block_group = %u, block_bitmap = %llu",
   360                :                            block_group, bitmap_blk);
   361              0 :                 return NULL;
   362                :         }
   363           4488 :         ext4_valid_block_bitmap(sb, desc, block_group, bh);
   364                :         /*
   365                :          * file system mounted not to panic on error,
   366                :          * continue with corrupt bitmap
   367                :          */
   368           4488 :         return bh;
   369                : }
   370                :
   371                : /**
   372                :  * ext4_add_groupblocks() -- Add given blocks to an existing group
   373                :  * @handle:                     handle to this transaction
   374                :  * @sb:                         super block
   375                :  * @block:                      start physcial block to add to the block group
   376                :  * @count:                      number of blocks to free
   377                :  *
   378                :  * This marks the blocks as free in the bitmap. We ask the
   379                :  * mballoc to reload the buddy after this by setting group
   380                :  * EXT4_GROUP_INFO_NEED_INIT_BIT flag
   381                :  */
   382                : void ext4_add_groupblocks(handle_t *handle, struct super_block *sb,
   383                :                           ext4_fsblk_t block, unsigned long count)
   384              0 : {
   385              0 :         struct buffer_head *bitmap_bh = NULL;
   386                :         struct buffer_head *gd_bh;
   387                :         ext4_group_t block_group;
   388                :         ext4_grpblk_t bit;
   389                :         unsigned int i;
   390                :         struct ext4_group_desc *desc;
   391                :         struct ext4_super_block *es;
   392                :         struct ext4_sb_info *sbi;
   393              0 :         int err = 0, ret, blk_free_count;
   394                :         ext4_grpblk_t blocks_freed;
   395                :         struct ext4_group_info *grp;
   396                :
   397              0 :         sbi = EXT4_SB(sb);
   398              0 :         es = sbi->s_es;
   399                :         ext4_debug("Adding block(s) %llu-%llu\n", block, block + count - 1);
   400                :
   401              0 :         ext4_get_group_no_and_offset(sb, block, &block_group, &bit);
   402              0 :         grp = ext4_get_group_info(sb, block_group);
   403                :         /*
   404                :          * Check to see if we are freeing blocks across a group
   405                :          * boundary.
   406                :          */
   407              0 :         if (bit + count > EXT4_BLOCKS_PER_GROUP(sb)) {
   408              0 :                 goto error_return;
   409                :         }
   410              0 :         bitmap_bh = ext4_read_block_bitmap(sb, block_group);
   411              0 :         if (!bitmap_bh)
   412              0 :                 goto error_return;
```

```
413                0 :          desc = ext4_get_group_desc(sb, block_group, &gd_bh);
414                0 :          if (!desc)
415                0 :                  goto error_return;
416                  :
417                0 :          if (in_range(ext4_block_bitmap(sb, desc), block, count) ||
418                  :              in_range(ext4_inode_bitmap(sb, desc), block, count) ||
419                  :              in_range(block, ext4_inode_table(sb, desc), sbi->s_itb_per_group) ||
420                  :              in_range(block + count - 1, ext4_inode_table(sb, desc),
421                  :                       sbi->s_itb_per_group)) {
422                0 :                  ext4_error(sb, __func__,
423                  :                             "Adding blocks in system zones - "
424                  :                             "Block = %llu, count = %lu",
425                  :                             block, count);
426                0 :                  goto error_return;
427                  :          }
428                  :
429                  :          /*
430                  :           * We are about to add blocks to the bitmap,
431                  :           * so we need undo access.
432                  :           */
433                  :          BUFFER_TRACE(bitmap_bh, "getting undo access");
434                0 :          err = ext4_journal_get_undo_access(handle, bitmap_bh);
435                0 :          if (err)
436                0 :                  goto error_return;
437                  :
438                  :          /*
439                  :           * We are about to modify some metadata.  Call the journal APIs
440                  :           * to unshare ->b_data if a currently-committing transaction is
441                  :           * using it
442                  :           */
443                  :          BUFFER_TRACE(gd_bh, "get_write_access");
444                0 :          err = ext4_journal_get_write_access(handle, gd_bh);
445                0 :          if (err)
446                0 :                  goto error_return;
447                  :          /*
448                  :           * make sure we don't allow a parallel init on other groups in the
449                  :           * same buddy cache
450                  :           */
451                0 :          down_write(&grp->alloc_sem);
452                0 :          for (i = 0, blocks_freed = 0; i < count; i++) {
453                  :                  BUFFER_TRACE(bitmap_bh, "clear bit");
454                0 :                  if (!ext4_clear_bit_atomic(ext4_group_lock_ptr(sb, block_group),
455                  :                                             bit + i, bitmap_bh->b_data)) {
456                0 :                          ext4_error(sb, __func__,
457                  :                                     "bit already cleared for block %llu",
458                  :                                     (ext4_fsblk_t)(block + i));
459                  :                          BUFFER_TRACE(bitmap_bh, "bit already cleared");
460                  :                  } else {
461                0 :                          blocks_freed++;
462                  :                  }
463                  :          }
464                0 :          ext4_lock_group(sb, block_group);
465                0 :          blk_free_count = blocks_freed + ext4_free_blks_count(sb, desc);
466                0 :          ext4_free_blks_set(sb, desc, blk_free_count);
467                0 :          desc->bg_checksum = ext4_group_desc_csum(sbi, block_group, desc);
468                0 :          ext4_unlock_group(sb, block_group);
469                0 :          percpu_counter_add(&sbi->s_freeblocks_counter, blocks_freed);
470                  :
471                0 :          if (sbi->s_log_groups_per_flex) {
472                0 :                  ext4_group_t flex_group = ext4_flex_group(sbi, block_group);
473                0 :                  atomic_add(blocks_freed,
474                  :                             &sbi->s_flex_groups[flex_group].free_blocks);
475                  :          }
476                  :          /*
477                  :           * request to reload the buddy with the
478                  :           * new bitmap information
479                  :           */
480                0 :          set_bit(EXT4_GROUP_INFO_NEED_INIT_BIT, &(grp->bb_state));
481                0 :          ext4_mb_update_group_info(grp, blocks_freed);
482                0 :          up_write(&grp->alloc_sem);
483                  :
```

```
                :         /* We dirtied the bitmap block */
                :         BUFFER_TRACE(bitmap_bh, "dirtied bitmap block");
              0 :         err = ext4_handle_dirty_metadata(handle, NULL, bitmap_bh);
                :
                :         /* And the group descriptor block */
                :         BUFFER_TRACE(gd_bh, "dirtied group descriptor block");
              0 :         ret = ext4_handle_dirty_metadata(handle, NULL, gd_bh);
              0 :         if (!err)
              0 :                 err = ret;
              0 :         sb->s_dirt = 1;
                :
              0 : error_return:
                :         brelse(bitmap_bh);
              0 :         ext4_std_error(sb, err);
                :         return;
                : }
                :
                : /**
                :  * ext4_free_blocks() -- Free given blocks and update quota
                :  * @handle:             handle for this transaction
                :  * @inode:              inode
                :  * @block:              start physical block to free
                :  * @count:              number of blocks to count
                :  * @metadata:           Are these metadata blocks
                :  */
                : void ext4_free_blocks(handle_t *handle, struct inode *inode,
                :                       ext4_fsblk_t block, unsigned long count,
                :                       int metadata)
          3896277 : {
                :         struct super_block *sb;
                :         unsigned long dquot_freed_blocks;
                :
                :         /* this isn't the right place to decide whether block is metadata
                :          * inode.c/extents.c knows better, but for safety ... */
          3896277 :         if (S_ISDIR(inode->i_mode) || S_ISLNK(inode->i_mode))
           655413 :                 metadata = 1;
                :
                :         /* We need to make sure we don't reuse
                :          * block released untill the transaction commit.
                :          * writeback mode have weak data consistency so
                :          * don't force data as metadata when freeing block
                :          * for writeback mode.
                :          */
          6921389 :         if (metadata == 0 && !ext4_should_writeback_data(inode))
          3025112 :                 metadata = 1;
                :
          3896277 :         sb = inode->i_sb;
                :
          3896277 :         ext4_mb_free_blocks(handle, inode, block, count,
                :                             metadata, &dquot_freed_blocks);
          3896273 :         if (dquot_freed_blocks)
          3896279 :                 vfs_dq_free_block(inode, dquot_freed_blocks);
                :         return;
                : }
                :
                : /**
                :  * ext4_has_free_blocks()
                :  * @sbi:        in-core super block structure.
                :  * @nblocks:    number of needed blocks
                :  *
                :  * Check if filesystem has nblocks free & available for allocation.
                :  * On success return 1, return 0 on failure.
                :  */
                : int ext4_has_free_blocks(struct ext4_sb_info *sbi, s64 nblocks)
        995215153 : {
                :         s64 free_blocks, dirty_blocks, root_blocks;
        995215153 :         struct percpu_counter *fbc = &sbi->s_freeblocks_counter;
        995215153 :         struct percpu_counter *dbc = &sbi->s_dirtyblocks_counter;
                :
        994424775 :         free_blocks  = percpu_counter_read_positive(fbc);
        995273491 :         dirty_blocks = percpu_counter_read_positive(dbc);
```

```
    555 1990546982 :         root_blocks = ext4_r_blocks_count(sbi->s_es);
    556            :
    557  995273491 :         if (free_blocks - (nblocks + root_blocks + dirty_blocks) <
    558            :                                                 EXT4_FREEBLOCKS_WATERMARK) {
    559       3100 :                 free_blocks  = percpu_counter_sum_positive(fbc);
    560       3100 :                 dirty_blocks = percpu_counter_sum_positive(dbc);
    561       3100 :                 if (dirty_blocks < 0) {
    562          0 :                         printk(KERN_CRIT "Dirty block accounting "
    563            :                                         "went wrong %lld\n",
    564            :                                         (long long)dirty_blocks);
    565            :                 }
    566            :         }
    567            :         /* Check whether we have space after
    568            :          * accounting for current dirty blocks & root reserved blocks.
    569            :          */
    570  993955845 :         if (free_blocks >= ((root_blocks + nblocks) + dirty_blocks))
    571  993955825 :                 return 1;
    572            :
    573            :         /* Hm, nope.  Are (enough) root reserved blocks available? */
    574         40 :         if (sbi->s_resuid == current_fsuid() ||
    575            :             ((sbi->s_resgid != 0) && in_group_p(sbi->s_resgid)) ||
    576            :             capable(CAP_SYS_RESOURCE)) {
    577         20 :                 if (free_blocks >= (nblocks + dirty_blocks))
    578          0 :                         return 1;
    579            :         }
    580            :
    581         20 :         return 0;
    582            : }
    583            :
    584            : int ext4_claim_free_blocks(struct ext4_sb_info *sbi,
    585            :                                                 s64 nblocks)
    586  995537659 : {
    587  995537659 :         if (ext4_has_free_blocks(sbi, nblocks)) {
    588  998316039 :                 percpu_counter_add(&sbi->s_dirtyblocks_counter, nblocks);
    589 1014731088 :                 return 0;
    590            :         } else
    591         10 :                 return -ENOSPC;
    592            : }
    593            :
    594            : /**
    595            :  * ext4_should_retry_alloc()
    596            :  * @sb:                 super block
    597            :  * @retries             number of attemps has been made
    598            :  *
    599            :  * ext4_should_retry_alloc() is called when ENOSPC is returned, and if
    600            :  * it is profitable to retry the operation, this function will wait
    601            :  * for the current or commiting transaction to complete, and then
    602            :  * return TRUE.
    603            :  *
    604            :  * if the total number of retries exceed three times, return FALSE.
    605            :  */
    606            : int ext4_should_retry_alloc(struct super_block *sb, int *retries)
    607         20 : {
    608         28 :         if (!ext4_has_free_blocks(EXT4_SB(sb), 1) ||
    609            :             (*retries)++ > 3 ||
    610            :             !EXT4_SB(sb)->s_journal)
    611         12 :                 return 0;
    612            :
    613          8 :         jbd_debug(1, "%s: retrying operation after ENOSPC\n", sb->s_id);
    614            :
    615          8 :         return jbd2_journal_force_commit_nested(EXT4_SB(sb)->s_journal);
    616            : }
    617            :
    618            : /*
    619            :  * ext4_new_meta_blocks() -- allocate block for meta data (indexing) blocks
    620            :  *
    621            :  * @handle:             handle to this transaction
    622            :  * @inode:              file inode
    623            :  * @goal:               given target block(filesystem wide)
    624            :  * @count:              pointer to total number of blocks needed
    625            :  * @errp:               error code
```

```
    626                 :  *
    627                 :  * Return 1st allocated block number on success, *count stores total account
    628                 :  * error stores in errp pointer
    629                 :  */
    630                 : ext4_fsblk_t ext4_new_meta_blocks(handle_t *handle, struct inode *inode,
    631                 :                                   ext4_fsblk_t goal, unsigned long *count, int *errp)
    632         1039640 : {
    633                 :         struct ext4_allocation_request ar;
    634                 :         ext4_fsblk_t ret;
    635                 :
    636                 :         memset(&ar, 0, sizeof(ar));
    637                 :         /* Fill with neighbour allocated blocks */
    638         1039632 :         ar.inode = inode;
    639         1039632 :         ar.goal = goal;
    640         1039632 :         ar.len = count ? *count : 1;
    641                 :
    642         1039632 :         ret = ext4_mb_new_blocks(handle, &ar, errp);
    643         1040693 :         if (count)
    644           77189 :                 *count = ar.len;
    645                 :
    646                 :         /*
    647                 :          * Account for the allocated meta blocks
    648                 :          */
    649         2081524 :         if (!(*errp) && EXT4_I(inode)->i_delalloc_reserved_flag) {
    650          642309 :                 spin_lock(&EXT4_I(inode)->i_block_reservation_lock);
    651          642338 :                 EXT4_I(inode)->i_allocated_meta_blocks += ar.len;
    652          642338 :                 spin_unlock(&EXT4_I(inode)->i_block_reservation_lock);
    653                 :         }
    654         1040752 :         return ret;
    655                 : }
    656                 :
    657                 : /**
    658                 :  * ext4_count_free_blocks() -- count filesystem free blocks
    659                 :  * @sb:         superblock
    660                 :  *
    661                 :  * Adds up the number of free blocks from each block group.
    662                 :  */
    663                 : ext4_fsblk_t ext4_count_free_blocks(struct super_block *sb)
    664             188 : {
    665                 :         ext4_fsblk_t desc_count;
    666                 :         struct ext4_group_desc *gdp;
    667                 :         ext4_group_t i;
    668             188 :         ext4_group_t ngroups = ext4_get_groups_count(sb);
    669                 : #ifdef EXT4FS_DEBUG
    670                 :         struct ext4_super_block *es;
    671                 :         ext4_fsblk_t bitmap_count;
    672                 :         unsigned int x;
    673                 :         struct buffer_head *bitmap_bh = NULL;
    674                 :
    675                 :         es = EXT4_SB(sb)->s_es;
    676                 :         desc_count = 0;
    677                 :         bitmap_count = 0;
    678                 :         gdp = NULL;
    679                 :
    680                 :         for (i = 0; i < ngroups; i++) {
    681                 :                 gdp = ext4_get_group_desc(sb, i, NULL);
    682                 :                 if (!gdp)
    683                 :                         continue;
    684                 :                 desc_count += ext4_free_blks_count(sb, gdp);
    685                 :                 brelse(bitmap_bh);
    686                 :                 bitmap_bh = ext4_read_block_bitmap(sb, i);
    687                 :                 if (bitmap_bh == NULL)
    688                 :                         continue;
    689                 :
    690                 :                 x = ext4_count_free(bitmap_bh, sb->s_blocksize);
    691                 :                 printk(KERN_DEBUG "group %u: stored = %d, counted = %u\n",
    692                 :                         i, ext4_free_blks_count(sb, gdp), x);
    693                 :                 bitmap_count += x;
    694                 :         }
    695                 :         brelse(bitmap_bh);
    696                 :         printk(KERN_DEBUG "ext4_count_free_blocks: stored = %llu"
```

```
697                :                 ", computed = %llu, %llu\n", ext4_free_blocks_count(es),
698                :                desc_count, bitmap_count);
699                :         return bitmap_count;
700                : #else
701            188 :         desc_count = 0;
702        3438268 :         for (i = 0; i < ngroups; i++) {
703        3438080 :                 gdp = ext4_get_group_desc(sb, i, NULL);
704        3438080 :                 if (!gdp)
705              0 :                         continue;
706        3438080 :                 desc_count += ext4_free_blks_count(sb, gdp);
707                :         }
708                :
709            188 :         return desc_count;
710                : #endif
711                : }
712                :
713                : static inline int test_root(ext4_group_t a, int b)
714                : {
715        5642177 :         int num = b;
716                :
717       39864264 :         while (a > num)
718       34222087 :                 num *= b;
719        5642177 :         return num == a;
720                : }
721                :
722                : static int ext4_group_sparse(ext4_group_t group)
723                : {
724        3661604 :         if (group <= 1)
725            236 :                 return 1;
726        3661368 :         if (!(group & 1))
727        1780107 :                 return 0;
728        5642177 :         return (test_root(group, 7) || test_root(group, 5) ||
729                :                 test_root(group, 3));
730                : }
731                :
732                : /**
733                :  *      ext4_bg_has_super - number of blocks used by the superblock in group
734                :  *      @sb: superblock for filesystem
735                :  *      @group: group number to check
736                :  *
737                :  *      Return the number of blocks used by the superblock (primary or backup)
738                :  *      in this group.  Currently this will be only 0 or 1.
739                :  */
740                : int ext4_bg_has_super(struct super_block *sb, ext4_group_t group)
741        3661604 : {
742        7323208 :         if (EXT4_HAS_RO_COMPAT_FEATURE(sb,
743                :                                        EXT4_FEATURE_RO_COMPAT_SPARSE_SUPER) &&
744                :                            !ext4_group_sparse(group))
745        3659372 :                 return 0;
746           2232 :         return 1;
747                : }
748                :
749                : static unsigned long ext4_bg_num_gdb_meta(struct super_block *sb,
750                :                                           ext4_group_t group)
751                : {
752              0 :         unsigned long metagroup = group / EXT4_DESC_PER_BLOCK(sb);
753              0 :         ext4_group_t first = metagroup * EXT4_DESC_PER_BLOCK(sb);
754              0 :         ext4_group_t last = first + EXT4_DESC_PER_BLOCK(sb) - 1;
755                :
756              0 :         if (group == first || group == first + 1 || group == last)
757              0 :                 return 1;
758              0 :         return 0;
759                : }
760                :
761                : static unsigned long ext4_bg_num_gdb_nometa(struct super_block *sb,
762                :                                             ext4_group_t group)
763                : {
764        1056476 :         return ext4_bg_has_super(sb, group) ? EXT4_SB(sb)->s_gdb_count : 0;
765                : }
766                :
767                : /**
```

```
768                 :  *      ext4_bg_num_gdb - number of blocks used by the group table in group
769                 :  *      @sb: superblock for filesystem
770                 :  *      @group: group number to check
771                 :  *
772                 :  *      Return the number of blocks used by the group descriptor table
773                 :  *      (primary or backup) in this group.  In the future there may be a
774                 :  *      different number of descriptor blocks in each group.
775                 :  */
776                 : unsigned long ext4_bg_num_gdb(struct super_block *sb, ext4_group_t group)
777         1055360 : {
778                 :         unsigned long first_meta_bg =
779         1055360 :                         le32_to_cpu(EXT4_SB(sb)->s_es->s_first_meta_bg);
780         1055360 :         unsigned long metagroup = group / EXT4_DESC_PER_BLOCK(sb);
781                 :
782         1055360 :         if (!EXT4_HAS_INCOMPAT_FEATURE(sb,EXT4_FEATURE_INCOMPAT_META_BG) ||
783                 :                         metagroup < first_meta_bg)
784         1055360 :                 return ext4_bg_num_gdb_nometa(sb, group);
785                 :
786               0 :         return ext4_bg_num_gdb_meta(sb,group);
787                 :
788                 : }
789                 :
```
