LCOV - code coverage report

			Found	Hit	Coverag
Current view:	directory - fs/ext4 - extents.c (source / functions)				
Test:	kernel_2_6_31_ext4_round_3.info	**Lines:**	1385	874	63.1 %
Date:	2009-10-24	**Functions:**	40	33	82.5 %

```
                :  1 /*
                :  2  * Copyright (c) 2003-2006, Cluster File Systems, Inc, info@clusterfs.com
                :  3  * Written by Alex Tomas <alex@clusterfs.com>
                :  4  *
                :  5  * Architecture independence:
                :  6  *   Copyright (c) 2005, Bull S.A.
                :  7  *   Written by Pierre Peiffer <pierre.peiffer@bull.net>
                :  8  *
                :  9  * This program is free software; you can redistribute it and/or modify
                :  10 * it under the terms of the GNU General Public License version 2 as
                :  11 * published by the Free Software Foundation.
                :  12 *
                :  13 * This program is distributed in the hope that it will be useful,
                :  14 * but WITHOUT ANY WARRANTY; without even the implied warranty of
                :  15 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
                :  16 * GNU General Public License for more details.
                :  17 *
                :  18 * You should have received a copy of the GNU General Public Licens
                :  19 * along with this program; if not, write to the Free Software
                :  20 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-
                :  21 */
                :  22
                :  23 /*
                :  24  * Extents support for EXT4
                :  25  *
                :  26  * TODO:
                :  27  *   - ext4*_error() should be used in some situations
                :  28  *   - analyze all BUG()/BUG_ON(), use -EIO where appropriate
                :  29  *   - smart tree reduction
                :  30  */
                :  31
                :  32 #include <linux/module.h>
                :  33 #include <linux/fs.h>
                :  34 #include <linux/time.h>
                :  35 #include <linux/jbd2.h>
                :  36 #include <linux/highuid.h>
                :  37 #include <linux/pagemap.h>
                :  38 #include <linux/quotaops.h>
                :  39 #include <linux/string.h>
                :  40 #include <linux/slab.h>
                :  41 #include <linux/falloc.h>
                :  42 #include <asm/uaccess.h>
                :  43 #include <linux/fiemap.h>
                :  44 #include "ext4_jbd2.h"
                :  45 #include "ext4_extents.h"
                :  46
                :  47
                :  48 /*
                :  49  * ext_pblock:
                :  50  * combine low and high parts of physical block number into ext4_fsblk_t
                :  51  */
                :  52 ext4_fsblk_t ext_pblock(struct ext4_extent *ex)
              0 :  53 {
                :  54         ext4_fsblk_t block;
                :  55
     2091889162 :  56         block = le32_to_cpu(ex->ee_start_lo);
     2091889162 :  57         block |= ((ext4_fsblk_t) le16_to_cpu(ex->ee_start_hi) << 31) << 1;
     2091889162 :  58         return block;
```

```
                :  }
                :
                :  /*
                :   * idx_pblock:
                :   * combine low and high parts of a leaf physical block number into ext4_fsblk_t
                :   */
                :  ext4_fsblk_t idx_pblock(struct ext4_extent_idx *ix)
              0 :  {
                :          ext4_fsblk_t block;
                :
      548444041 :          block = le32_to_cpu(ix->ei_leaf_lo);
      548444041 :          block |= ((ext4_fsblk_t) le16_to_cpu(ix->ei_leaf_hi) << 31) << 1;
      548444041 :          return block;
                :  }
                :
                :  /*
                :   * ext4_ext_store_pblock:
                :   * stores a large physical block number into an extent struct,
                :   * breaking it into parts
                :   */
                :  void ext4_ext_store_pblock(struct ext4_extent *ex, ext4_fsblk_t pb)
              0 :  {
     1305835123 :          ex->ee_start_lo = cpu_to_le32((unsigned long) (pb & 0xffffffff));
     1305835123 :          ex->ee_start_hi = cpu_to_le16((unsigned long) ((pb >> 31) >> 1) & 0xffff);
              0 :  }
                :
                :  /*
                :   * ext4_idx_store_pblock:
                :   * stores a large physical block number into an index struct,
                :   * breaking it into parts
                :   */
                :  static void ext4_idx_store_pblock(struct ext4_extent_idx *ix, ext4_fsblk_t pb)
                :  {
         964121 :          ix->ei_leaf_lo = cpu_to_le32((unsigned long) (pb & 0xffffffff));
         964121 :          ix->ei_leaf_hi = cpu_to_le16((unsigned long) ((pb >> 31) >> 1) & 0xffff);
                :  }
                :
                :  static int ext4_ext_journal_restart(handle_t *handle, int needed)
                :  {
                :          int err;
                :
        3680516 :          if (!ext4_handle_valid(handle))
              0 :                  return 0;
        3680516 :          if (handle->h_buffer_credits > needed)
        3680516 :                  return 0;
              0 :          err = ext4_journal_extend(handle, needed);
              0 :          if (err <= 0)
              0 :                  return err;
              0 :          return ext4_journal_restart(handle, needed);
                :  }
                :
                :  /*
                :   * could return:
                :   *  - EROFS
                :   *  - ENOMEM
                :   */
                :  static int ext4_ext_get_access(handle_t *handle, struct inode *inode,
                :                                   struct ext4_ext_path *path)
                :  {
      227039169 :          if (path->p_bh) {
                :                  /* path points to block */
      175386365 :                  return ext4_journal_get_write_access(handle, path->p_bh);
                :          }
                :          /* path points to leaf/index in inode body */
                :          /* we use in-core data, no need to protect them */
       51652804 :          return 0;
                :  }
                :
                :  /*
                :   * could return:
                :   *  - EROFS
```

```
130                 :  *  - ENOMEM
131                 :  *  - EIO
132                 :  */
133                 : static int ext4_ext_dirty(handle_t *handle, struct inode *inode,
134                 :                           struct ext4_ext_path *path)
135       227298159 : {
136                 :         int err;
137       227298159 :         if (path->p_bh) {
138                 :                 /* path points to block */
139       175625595 :                 err = ext4_handle_dirty_metadata(handle, inode, path->p_bh);
140                 :         } else {
141                 :                 /* path points to leaf/index in inode body */
142        51672564 :                 err = ext4_mark_inode_dirty(handle, inode);
143                 :         }
144       227523476 :         return err;
145                 : }
146                 :
147                 : static ext4_fsblk_t ext4_ext_find_goal(struct inode *inode,
148                 :                               struct ext4_ext_path *path,
149                 :                               ext4_lblk_t block)
150       220261118 : {
151       220261118 :         struct ext4_inode_info *ei = EXT4_I(inode);
152                 :         ext4_fsblk_t bg_start;
153                 :         ext4_fsblk_t last_block;
154                 :         ext4_grpblk_t colour;
155                 :         ext4_group_t block_group;
156       440522236 :         int flex_size = ext4_flex_bg_size(EXT4_SB(inode->i_sb));
157                 :         int depth;
158                 :
159       220261118 :         if (path) {
160                 :                 struct ext4_extent *ex;
161       220261110 :                 depth = path->p_depth;
162                 :
163                 :                 /* try to predict block placement */
164       220261110 :                 ex = path[depth].p_ext;
165       220261110 :                 if (ex)
166       211050079 :                         return ext_pblock(ex)+(block-le32_to_cpu(ex->ee_block));
167                 :
168                 :                 /* it looks like index is empty;
169                 :                  * try to find starting block from index itself */
170         9211031 :                 if (path[depth].p_bh)
171               0 :                         return path[depth].p_bh->b_blocknr;
172                 :         }
173                 :
174                 :         /* OK. use inode's group */
175         9211039 :         block_group = ei->i_block_group;
176         9211039 :         if (flex_size >= EXT4_FLEX_SIZE_DIR_ALLOC_SCHEME) {
177                 :                 /*
178                 :                  * If there are at least EXT4_FLEX_SIZE_DIR_ALLOC_SCHEME
179                 :                  * block groups per flexgroup, reserve the first block
180                 :                  * group for directories and special files.  Regular
181                 :                  * files will start at the second block group.  This
182                 :                  * tends to speed up directory access and improves
183                 :                  * fsck times.
184                 :                  */
185         9211041 :                 block_group &= ~(flex_size-1);
186         9211041 :                 if (S_ISREG(inode->i_mode))
187         8513414 :                         block_group++;
188                 :         }
189        27633117 :         bg_start = (block_group * EXT4_BLOCKS_PER_GROUP(inode->i_sb)) +
190                 :                 le32_to_cpu(EXT4_SB(inode->i_sb)->s_es->s_first_data_block);
191        27633117 :         last_block = ext4_blocks_count(EXT4_SB(inode->i_sb)->s_es) - 1;
192                 :
193                 :         /*
194                 :          * If we are doing delayed allocation, we don't need take
195                 :          * colour into account.
196                 :          */
197        18422078 :         if (test_opt(inode->i_sb, DELALLOC))
198         6778519 :                 return bg_start;
199                 :
200         4865040 :         if (bg_start + EXT4_BLOCKS_PER_GROUP(inode->i_sb) <= last_block)
```

```
      4865040 :                 colour = (current->pid % 16) *
              :                         (EXT4_BLOCKS_PER_GROUP(inode->i_sb) / 16);
              :         else
            0 :                 colour = (current->pid % 16) * ((last_block - bg_start) / 16);
      2432520 :         return bg_start + colour + block;
              : }
              :
              : /*
              :  * Allocation for a meta data block
              :  */
              : static ext4_fsblk_t
              : ext4_ext_new_meta_block(handle_t *handle, struct inode *inode,
              :                         struct ext4_ext_path *path,
              :                         struct ext4_extent *ex, int *err)
              : {
              :         ext4_fsblk_t goal, newblock;
              :
       962501 :         goal = ext4_ext_find_goal(inode, path, le32_to_cpu(ex->ee_block));
       963017 :         newblock = ext4_new_meta_blocks(handle, inode, goal, NULL, err);
       963339 :         return newblock;
              : }
              :
              : static int ext4_ext_space_block(struct inode *inode)
              : {
              :         int size;
              :
    791200248 :         size = (inode->i_sb->s_blocksize - sizeof(struct ext4_extent_header))
              :                         / sizeof(struct ext4_extent);
              : #ifdef AGGRESSIVE_TEST
              :         if (size > 6)
              :                 size = 6;
              : #endif
    791200248 :         return size;
              : }
              :
              : static int ext4_ext_space_block_idx(struct inode *inode)
              : {
              :         int size;
              :
    789880828 :         size = (inode->i_sb->s_blocksize - sizeof(struct ext4_extent_header))
              :                         / sizeof(struct ext4_extent_idx);
              : #ifdef AGGRESSIVE_TEST
              :         if (size > 5)
              :                 size = 5;
              : #endif
    789880828 :         return size;
              : }
              :
              : static int ext4_ext_space_root(struct inode *inode)
              : {
              :         int size;
              :
       15412162 :         size = sizeof(EXT4_I(inode)->i_data);
       15412162 :         size -= sizeof(struct ext4_extent_header);
       15412162 :         size /= sizeof(struct ext4_extent);
              : #ifdef AGGRESSIVE_TEST
              :         if (size > 3)
              :                 size = 3;
              : #endif
       15412162 :         return size;
              : }
              :
              : static int ext4_ext_space_root_idx(struct inode *inode)
              : {
              :         int size;
              :
    791732160 :         size = sizeof(EXT4_I(inode)->i_data);
    791732160 :         size -= sizeof(struct ext4_extent_header);
    791732160 :         size /= sizeof(struct ext4_extent_idx);
              : #ifdef AGGRESSIVE_TEST
              :         if (size > 4)
```

```
       :                 size = 4;
       : #endif
 791732160 :         return size;
       : }
       :
       : /*
       :  * Calculate the number of metadata blocks needed
       :  * to allocate @blocks
       :  * Worse case is one block per extent
       :  */
       : int ext4_ext_calc_metadata_amount(struct inode *inode, int blocks)
 789880823 : {
       :         int lcap, icap, rcap, leafs, idxs, num;
 789880823 :         int newextents = blocks;
       :
 789880823 :         rcap = ext4_ext_space_root_idx(inode);
 789880823 :         lcap = ext4_ext_space_block(inode);
 789880823 :         icap = ext4_ext_space_block_idx(inode);
       :
       :         /* number of new leaf blocks needed */
 789880823 :         num = leafs = (newextents + lcap - 1) / lcap;
       :
       :         /*
       :          * Worse case, we need separate index block(s)
       :          * to link all new leaf blocks
       :          */
 789880823 :         idxs = (leafs + icap - 1) / icap;
       :         do {
 789880823 :                 num += idxs;
 789880823 :                 idxs = (idxs + icap - 1) / icap;
 789880823 :         } while (idxs > rcap);
       :
 789880823 :         return num;
       : }
       :
       : static int
       : ext4_ext_max_entries(struct inode *inode, int depth)
       : {
       :         int max;
       :
   5780452 :         if (depth == ext_depth(inode)) {
   5424113 :                 if (depth == 0)
   4536755 :                         max = ext4_ext_space_root(inode);
       :                 else
    887358 :                         max = ext4_ext_space_root_idx(inode);
       :         } else {
    356339 :                 if (depth == 0)
    356337 :                         max = ext4_ext_space_block(inode);
       :                 else
         2 :                         max = ext4_ext_space_block_idx(inode);
       :         }
       :
   5780452 :         return max;
       : }
       :
       : static int ext4_valid_extent(struct inode *inode, struct ext4_extent *ext)
       : {
  10224871 :         ext4_fsblk_t block = ext_pblock(ext);
  10224871 :         int len = ext4_ext_get_actual_len(ext);
       :
  20449742 :         return ext4_data_block_valid(EXT4_SB(inode->i_sb), block, len);
       : }
       :
       : static int ext4_valid_extent_idx(struct inode *inode,
       :                                  struct ext4_extent_idx *ext_idx)
       : {
    887397 :         ext4_fsblk_t block = idx_pblock(ext_idx);
       :
   1774794 :         return ext4_data_block_valid(EXT4_SB(inode->i_sb), block, 1);
       : }
       :
```

```
    343                 : static int ext4_valid_extent_entries(struct inode *inode,
    344                 :                                      struct ext4_extent_header *eh,
    345                 :                                      int depth)
    346                 : {
    347                 :         struct ext4_extent *ext;
    348                 :         struct ext4_extent_idx *ext_idx;
    349                 :         unsigned short entries;
    350         5780451 :         if (eh->eh_entries == 0)
    351               1 :                 return 1;
    352                 :
    353         5780450 :         entries = le16_to_cpu(eh->eh_entries);
    354                 :
    355         5780450 :         if (depth == 0) {
    356                 :                 /* leaf entries */
    357         4893090 :                 ext = EXT_FIRST_EXTENT(eh);
    358        15117961 :                 while (entries) {
    359        10224871 :                         if (!ext4_valid_extent(inode, ext))
    360               0 :                                 return 0;
    361        10224871 :                         ext++;
    362        10224871 :                         entries--;
    363                 :                 }
    364                 :         } else {
    365          887360 :                 ext_idx = EXT_FIRST_INDEX(eh);
    366         1774757 :                 while (entries) {
    367          887397 :                         if (!ext4_valid_extent_idx(inode, ext_idx))
    368               0 :                                 return 0;
    369          887397 :                         ext_idx++;
    370          887397 :                         entries--;
    371                 :                 }
    372                 :         }
    373         5780450 :         return 1;
    374                 : }
    375                 :
    376                 : static int __ext4_ext_check(const char *function, struct inode *inode,
    377                 :                             struct ext4_extent_header *eh,
    378                 :                             int depth)
    379         5780447 : {
    380                 :         const char *error_msg;
    381         5780447 :         int max = 0;
    382                 :
    383         5780447 :         if (unlikely(eh->eh_magic != EXT4_EXT_MAGIC)) {
    384               0 :                 error_msg = "invalid magic";
    385               0 :                 goto corrupted;
    386                 :         }
    387         5780451 :         if (unlikely(le16_to_cpu(eh->eh_depth) != depth)) {
    388               0 :                 error_msg = "unexpected eh_depth";
    389               0 :                 goto corrupted;
    390                 :         }
    391         5780451 :         if (unlikely(eh->eh_max == 0)) {
    392               0 :                 error_msg = "invalid eh_max";
    393               0 :                 goto corrupted;
    394                 :         }
    395         5780452 :         max = ext4_ext_max_entries(inode, depth);
    396         5780452 :         if (unlikely(le16_to_cpu(eh->eh_max) > max)) {
    397               0 :                 error_msg = "too large eh_max";
    398               0 :                 goto corrupted;
    399                 :         }
    400         5780454 :         if (unlikely(le16_to_cpu(eh->eh_entries) > le16_to_cpu(eh->eh_max))) {
    401               0 :                 error_msg = "invalid eh_entries";
    402               0 :                 goto corrupted;
    403                 :         }
    404         5780451 :         if (!ext4_valid_extent_entries(inode, eh, depth)) {
    405               0 :                 error_msg = "invalid extent entries";
    406               0 :                 goto corrupted;
    407                 :         }
    408         5780451 :         return 0;
    409                 :
    410               0 : corrupted:
    411               0 :         ext4_error(inode->i_sb, function,
    412                 :                    "bad header/extent in inode #%lu: %s - magic %x, "
    413                 :                    "entries %u, max %u(%u), depth %u(%u)",
```

```
                :                         inode->i_ino, error_msg, le16_to_cpu(eh->eh_magic),
                :                         le16_to_cpu(eh->eh_entries), le16_to_cpu(eh->eh_max),
                :                         max, le16_to_cpu(eh->eh_depth), depth);
                :
              0 :         return -EIO;
                : }
                :
                : #define ext4_ext_check(inode, eh, depth)        \
                :         __ext4_ext_check(__func__, inode, eh, depth)
                :
                : int ext4_ext_check_inode(struct inode *inode)
        3761217 : {
        7522434 :         return ext4_ext_check(inode, ext_inode_hdr(inode), ext_depth(inode));
                : }
                :
                : #ifdef EXT_DEBUG
                : static void ext4_ext_show_path(struct inode *inode, struct ext4_ext_path *path)
                : {
                :         int k, l = path->p_depth;
                :
                :         ext_debug("path:");
                :         for (k = 0; k <= l; k++, path++) {
                :                 if (path->p_idx) {
                :                   ext_debug("  %d->%llu", le32_to_cpu(path->p_idx->ei_block),
                :                             idx_pblock(path->p_idx));
                :                 } else if (path->p_ext) {
                :                         ext_debug("  %d:%d:%llu ",
                :                                   le32_to_cpu(path->p_ext->ee_block),
                :                                   ext4_ext_get_actual_len(path->p_ext),
                :                                   ext_pblock(path->p_ext));
                :                 } else
                :                         ext_debug("  []");
                :         }
                :         ext_debug("\n");
                : }
                :
                : static void ext4_ext_show_leaf(struct inode *inode, struct ext4_ext_path *path)
                : {
                :         int depth = ext_depth(inode);
                :         struct ext4_extent_header *eh;
                :         struct ext4_extent *ex;
                :         int i;
                :
                :         if (!path)
                :                 return;
                :
                :         eh = path[depth].p_hdr;
                :         ex = EXT_FIRST_EXTENT(eh);
                :
                :         for (i = 0; i < le16_to_cpu(eh->eh_entries); i++, ex++) {
                :                 ext_debug("%d:%d:%llu ", le32_to_cpu(ex->ee_block),
                :                           ext4_ext_get_actual_len(ex), ext_pblock(ex));
                :         }
                :         ext_debug("\n");
                : }
                : #else
                : #define ext4_ext_show_path(inode, path)
                : #define ext4_ext_show_leaf(inode, path)
                : #endif
                :
                : void ext4_ext_drop_refs(struct ext4_ext_path *path)
      494516907 : {
      494516907 :         int depth = path->p_depth;
                :         int i;
                :
     1538322680 :         for (i = 0; i <= depth; i++, path++)
     1042401857 :                 if (path->p_bh) {
      546686087 :                         brelse(path->p_bh);
      548090003 :                         path->p_bh = NULL;
                :                 }
      495920823 : }
```

```
485                 :
486                 : /*
487                 :  * ext4_ext_binsearch_idx:
488                 :  * binary search for the closest index of the given block
489                 :  * the header must be checked before calling this
490                 :  */
491                 : static void
492                 : ext4_ext_binsearch_idx(struct inode *inode,
493                 :                        struct ext4_ext_path *path, ext4_lblk_t block)
494                 : {
495       547125128 :         struct ext4_extent_header *eh = path->p_hdr;
496                 :         struct ext4_extent_idx *r, *l, *m;
497                 :
498                 :
499                 :         ext_debug("binsearch for %u(idx):  ", block);
500                 :
501       547125128 :         l = EXT_FIRST_INDEX(eh) + 1;
502       547125128 :         r = EXT_LAST_INDEX(eh);
503      1248704597 :         while (l <= r) {
504       701579469 :                 m = l + (r - l) / 2;
505       701579469 :                 if (block < le32_to_cpu(m->ei_block))
506         1176106 :                         r = m - 1;
507                 :                 else
508       700403363 :                         l = m + 1;
509                 :                 ext_debug("%p(%u):%p(%u):%p(%u) ", l, le32_to_cpu(l->ei_block),
510                 :                                 m, le32_to_cpu(m->ei_block),
511                 :                                 r, le32_to_cpu(r->ei_block));
512                 :         }
513                 :
514       547125128 :         path->p_idx = l - 1;
515                 :         ext_debug("  -> %d->%lld ", le32_to_cpu(path->p_idx->ei_block),
516                 :                   idx_pblock(path->p_idx));
517                 :
518                 : #ifdef CHECK_BINSEARCH
519                 :         {
520                 :                 struct ext4_extent_idx *chix, *ix;
521                 :                 int k;
522                 :
523                 :                 chix = ix = EXT_FIRST_INDEX(eh);
524                 :                 for (k = 0; k < le16_to_cpu(eh->eh_entries); k++, ix++) {
525                 :                   if (k != 0 &&
526                 :                       le32_to_cpu(ix->ei_block) <= le32_to_cpu(ix[-1].ei_block)) {
527                 :                                 printk(KERN_DEBUG "k=%d, ix=0x%p, "
528                 :                                         "first=0x%p\n", k,
529                 :                                         ix, EXT_FIRST_INDEX(eh));
530                 :                                 printk(KERN_DEBUG "%u <= %u\n",
531                 :                                         le32_to_cpu(ix->ei_block),
532                 :                                         le32_to_cpu(ix[-1].ei_block));
533                 :                           }
534                 :                           BUG_ON(k && le32_to_cpu(ix->ei_block)
535                 :                                            <= le32_to_cpu(ix[-1].ei_block));
536                 :                           if (block < le32_to_cpu(ix->ei_block))
537                 :                                   break;
538                 :                           chix = ix;
539                 :                 }
540                 :                 BUG_ON(chix != path->p_idx);
541                 :         }
542                 : #endif
543                 :
544                 : }
545                 :
546                 : /*
547                 :  * ext4_ext_binsearch:
548                 :  * binary search for closest extent of the given block
549                 :  * the header must be checked before calling this
550                 :  */
551                 : static void
552                 : ext4_ext_binsearch(struct inode *inode,
553                 :                 struct ext4_ext_path *path, ext4_lblk_t block)
554                 : {
555       495652917 :         struct ext4_extent_header *eh = path->p_hdr;
```

```
       :         struct ext4_extent *r, *l, *m;
       :
495652917 :         if (eh->eh_entries == 0) {
       :                 /*
       :                  * this leaf is empty:
       :                  * we get such a leaf in split/add case
       :                  */
       :                 return;
       :         }
       :
       :         ext_debug("binsearch for %u:  ", block);
       :
476932430 :         l = EXT_FIRST_EXTENT(eh) + 1;
476932430 :         r = EXT_LAST_EXTENT(eh);
       :
-1335157912 :         while (l <= r) {
-1812090342 :                 m = l + (r - l) / 2;
-1812090342 :                 if (block < le32_to_cpu(m->ee_block))
 106496729 :                         r = m - 1;
       :                 else
-1918587071 :                         l = m + 1;
       :                 ext_debug("%p(%u):%p(%u):%p(%u) ", l, le32_to_cpu(l->ee_block),
       :                                 m, le32_to_cpu(m->ee_block),
       :                                 r, le32_to_cpu(r->ee_block));
       :         }
       :
476932430 :         path->p_ext = l - 1;
       :         ext_debug("  -> %d:%llu:%d ",
       :                         le32_to_cpu(path->p_ext->ee_block),
       :                         ext_pblock(path->p_ext),
       :                         ext4_ext_get_actual_len(path->p_ext));
       :
       : #ifdef CHECK_BINSEARCH
       :         {
       :                 struct ext4_extent *chex, *ex;
       :                 int k;
       :
       :                 chex = ex = EXT_FIRST_EXTENT(eh);
       :                 for (k = 0; k < le16_to_cpu(eh->eh_entries); k++, ex++) {
       :                         BUG_ON(k && le32_to_cpu(ex->ee_block)
       :                                         <= le32_to_cpu(ex[-1].ee_block));
       :                         if (block < le32_to_cpu(ex->ee_block))
       :                                 break;
       :                         chex = ex;
       :                 }
       :                 BUG_ON(chex != path->p_ext);
       :         }
       : #endif
       :
       : }
       :
       : int ext4_ext_tree_init(handle_t *handle, struct inode *inode)
  9212522 : {
       :         struct ext4_extent_header *eh;
       :
  9212522 :         eh = ext_inode_hdr(inode);
  9212522 :         eh->eh_depth = 0;
  9212522 :         eh->eh_entries = 0;
  9212522 :         eh->eh_magic = EXT4_EXT_MAGIC;
  9212522 :         eh->eh_max = cpu_to_le16(ext4_ext_space_root(inode));
  9212522 :         ext4_mark_inode_dirty(handle, inode);
       :         ext4_ext_invalidate_cache(inode);
  9212522 :         return 0;
       : }
       :
       : struct ext4_ext_path *
       : ext4_ext_find_extent(struct inode *inode, ext4_lblk_t block,
       :                                         struct ext4_ext_path *path)
493910295 : {
       :         struct ext4_extent_header *eh;
       :         struct buffer_head *bh;
```

```
   627  493910295 :          short int depth, i, ppos = 0, alloc = 0;
   628            :
   629  493910295 :          eh = ext_inode_hdr(inode);
   630  493910295 :          depth = ext_depth(inode);
   631            :
   632            :          /* account possible depth increase */
   633  493910295 :          if (!path) {
   634  491187343 :                  path = kzalloc(sizeof(struct ext4_ext_path) * (depth + 2),
   635            :                                 GFP_NOFS);
   636  491500839 :                  if (!path)
   637          0 :                          return ERR_PTR(-ENOMEM);
   638  491500839 :                  alloc = 1;
   639            :          }
   640  494223791 :          path[0].p_hdr = eh;
   641  494223791 :          path[0].p_bh = NULL;
   642            :
   643  494223791 :          i = depth;
   644            :          /* walk through the tree */
   645 1537001836 :          while (i) {
   646  547125128 :                  int need_to_validate = 0;
   647            :
   648            :                  ext_debug("depth %d: num %d, max %d\n",
   649            :                            ppos, le16_to_cpu(eh->eh_entries), le16_to_cpu(eh->eh_max));
   650            :
   651  547125128 :                  ext4_ext_binsearch_idx(inode, path + ppos, block);
   652 1094250256 :                  path[ppos].p_block = idx_pblock(path[ppos].p_idx);
   653  547125128 :                  path[ppos].p_depth = i;
   654  547125128 :                  path[ppos].p_ext = NULL;
   655            :
   656 1094941820 :                  bh = sb_getblk(inode->i_sb, path[ppos].p_block);
   657  547816692 :                  if (unlikely(!bh))
   658          0 :                          goto err;
   659  547442952 :                  if (!bh_uptodate_or_lock(bh)) {
   660     140577 :                          if (bh_submit_read(bh) < 0) {
   661            :                                  put_bh(bh);
   662            :                                  goto err;
   663            :                          }
   664            :                          /* validate the extent entries */
   665     140575 :                          need_to_validate = 1;
   666            :                  }
   667  548533040 :                  eh = ext_block_hdr(bh);
   668  548533040 :                  ppos++;
   669  548533040 :                  BUG_ON(ppos > depth);
   670  548554254 :                  path[ppos].p_bh = bh;
   671  548554254 :                  path[ppos].p_hdr = eh;
   672  548554254 :                  i--;
   673            :
   674  548554254 :                  if (need_to_validate && ext4_ext_check(inode, eh, i))
   675          0 :                          goto err;
   676            :          }
   677            :
   678  495652917 :          path[ppos].p_depth = i;
   679  495652917 :          path[ppos].p_ext = NULL;
   680  495652917 :          path[ppos].p_idx = NULL;
   681            :
   682            :          /* find extent */
   683  495652917 :          ext4_ext_binsearch(inode, path + ppos, block);
   684            :          /* if not an empty leaf */
   685  495652917 :          if (path[ppos].p_ext)
   686  954194520 :                  path[ppos].p_block = ext_pblock(path[ppos].p_ext);
   687            :
   688            :          ext4_ext_show_path(inode, path);
   689            :
   690  495652917 :          return path;
   691            :
   692          0 : err:
   693          0 :          ext4_ext_drop_refs(path);
   694          0 :          if (alloc)
   695          0 :                  kfree(path);
   696          0 :          return ERR_PTR(-EIO);
   697            : }
```

```
                :
                : /*
                :  * ext4_ext_insert_index:
                :  * insert new index [@logical;@ptr] into the block at @curp;
                :  * check where to insert: before @curp or after @curp
                :  */
                : static int ext4_ext_insert_index(handle_t *handle, struct inode *inode,
                :                                  struct ext4_ext_path *curp,
                :                                  int logical, ext4_fsblk_t ptr)
           142 : {
                :         struct ext4_extent_idx *ix;
                :         int len, err;
                :
           142 :         err = ext4_ext_get_access(handle, inode, curp);
           142 :         if (err)
             0 :                 return err;
                :
           142 :         BUG_ON(logical == le32_to_cpu(curp->p_idx->ei_block));
           142 :         len = EXT_MAX_INDEX(curp->p_hdr) - curp->p_idx;
           142 :         if (logical > le32_to_cpu(curp->p_idx->ei_block)) {
                :                 /* insert after */
           142 :                 if (curp->p_idx != EXT_LAST_INDEX(curp->p_hdr)) {
             0 :                         len = (len - 1) * sizeof(struct ext4_extent_idx);
             0 :                         len = len < 0 ? 0 : len;
                :                         ext_debug("insert new index %d after: %llu. "
                :                                         "move %d from 0x%p to 0x%p\n",
                :                                         logical, ptr, len,
                :                                         (curp->p_idx + 1), (curp->p_idx + 2));
             0 :                         memmove(curp->p_idx + 2, curp->p_idx + 1, len);
                :                 }
           142 :                 ix = curp->p_idx + 1;
                :         } else {
                :                 /* insert before */
             0 :                 len = len * sizeof(struct ext4_extent_idx);
             0 :                 len = len < 0 ? 0 : len;
                :                 ext_debug("insert new index %d before: %llu. "
                :                                 "move %d from 0x%p to 0x%p\n",
                :                                 logical, ptr, len,
                :                                 curp->p_idx, (curp->p_idx + 1));
             0 :                 memmove(curp->p_idx + 1, curp->p_idx, len);
             0 :                 ix = curp->p_idx;
                :         }
                :
           142 :         ix->ei_block = cpu_to_le32(logical);
                :         ext4_idx_store_pblock(ix, ptr);
           142 :         le16_add_cpu(&curp->p_hdr->eh_entries, 1);
                :
           142 :         BUG_ON(le16_to_cpu(curp->p_hdr->eh_entries)
                :                              > le16_to_cpu(curp->p_hdr->eh_max));
           142 :         BUG_ON(ix > EXT_LAST_INDEX(curp->p_hdr));
                :
           142 :         err = ext4_ext_dirty(handle, inode, curp);
           142 :         ext4_std_error(inode->i_sb, err);
                :
           142 :         return err;
                : }
                :
                : /*
                :  * ext4_ext_split:
                :  * inserts new subtree into the path, using free index entry
                :  * at depth @at:
                :  * - allocates all needed blocks (new leaf and all intermediate index blocks)
                :  * - makes decision where to split
                :  * - moves remaining extents and index entries (right to the split point)
                :  *   into the newly allocated blocks
                :  * - initializes subtree
                :  */
                : static int ext4_ext_split(handle_t *handle, struct inode *inode,
                :                           struct ext4_ext_path *path,
                :                           struct ext4_extent *newext, int at)
           142 : {
```

```
769      142 :          struct buffer_head *bh = NULL;
770      142 :          int depth = ext_depth(inode);
771          :          struct ext4_extent_header *neh;
772          :          struct ext4_extent_idx *fidx;
773          :          struct ext4_extent *ex;
774      142 :          int i = at, k, m, a;
775          :          ext4_fsblk_t newblock, oldblock;
776          :          __le32 border;
777      142 :          ext4_fsblk_t *ablocks = NULL; /* array of allocated blocks */
778      142 :          int err = 0;
779          :
780          :          /* make decision: where to split? */
781          :          /* FIXME: now decision is simplest: at current extent */
782          :
783          :          /* if current leaf will be split, then we should use
784          :           * border from split point */
785      142 :          BUG_ON(path[depth].p_ext > EXT_MAX_EXTENT(path[depth].p_hdr));
786      142 :          if (path[depth].p_ext != EXT_MAX_EXTENT(path[depth].p_hdr)) {
787        0 :                  border = path[depth].p_ext[1].ee_block;
788          :                  ext_debug("leaf will be split."
789          :                                  " next leaf starts at %d\n",
790          :                                    le32_to_cpu(border));
791          :          } else {
792      142 :                  border = newext->ee_block;
793          :                  ext_debug("leaf will be added."
794          :                                  " next leaf starts at %d\n",
795          :                                  le32_to_cpu(border));
796          :          }
797          :
798          :          /*
799          :           * If error occurs, then we break processing
800          :           * and mark filesystem read-only. index won't
801          :           * be inserted and tree will be in consistent
802          :           * state. Next mount will repair buffers too.
803          :           */
804          :
805          :          /*
806          :           * Get array to track all allocated blocks.
807          :           * We need this to handle errors and free blocks
808          :           * upon them.
809          :           */
810      142 :          ablocks = kzalloc(sizeof(ext4_fsblk_t) * depth, GFP_NOFS);
811      142 :          if (!ablocks)
812        0 :                  return -ENOMEM;
813          :
814          :          /* allocate all needed blocks */
815          :          ext_debug("allocate %d blocks for indexes/leaf\n", depth - at);
816      284 :          for (a = 0; a < depth - at; a++) {
817      142 :                  newblock = ext4_ext_new_meta_block(handle, inode, path,
818          :                                                     newext, &err);
819      142 :                  if (newblock == 0)
820        0 :                          goto cleanup;
821      142 :                  ablocks[a] = newblock;
822          :          }
823          :
824          :          /* initialize new leaf */
825      142 :          newblock = ablocks[--a];
826      142 :          BUG_ON(newblock == 0);
827      284 :          bh = sb_getblk(inode->i_sb, newblock);
828      142 :          if (!bh) {
829        0 :                  err = -EIO;
830        0 :                  goto cleanup;
831          :          }
832      142 :          lock_buffer(bh);
833          :
834      142 :          err = ext4_journal_get_create_access(handle, bh);
835      142 :          if (err)
836        0 :                  goto cleanup;
837          :
838      142 :          neh = ext_block_hdr(bh);
839      142 :          neh->eh_entries = 0;
```

```
840      142 :          neh->eh_max = cpu_to_le16(ext4_ext_space_block(inode));
841      142 :          neh->eh_magic = EXT4_EXT_MAGIC;
842      142 :          neh->eh_depth = 0;
843      142 :          ex = EXT_FIRST_EXTENT(neh);
844          :
845          :          /* move remainder of path[depth] to the new leaf */
846      142 :          BUG_ON(path[depth].p_hdr->eh_entries != path[depth].p_hdr->eh_max);
847          :          /* start copy from next extent */
848          :          /* TODO: we could do it by single memmove */
849      142 :          m = 0;
850      142 :          path[depth].p_ext++;
851      284 :          while (path[depth].p_ext <=
852          :                                  EXT_MAX_EXTENT(path[depth].p_hdr)) {
853          :                  ext_debug("move %d:%llu:%d in new leaf %llu\n",
854          :                                  le32_to_cpu(path[depth].p_ext->ee_block),
855          :                                  ext_pblock(path[depth].p_ext),
856          :                                  ext4_ext_get_actual_len(path[depth].p_ext),
857          :                                  newblock);
858          :                  /*memmove(ex++, path[depth].p_ext++,
859          :                                  sizeof(struct ext4_extent));
860          :                  neh->eh_entries++;*/
861        0 :                  path[depth].p_ext++;
862        0 :                  m++;
863          :          }
864      142 :          if (m) {
865        0 :                  memmove(ex, path[depth].p_ext-m, sizeof(struct ext4_extent)*m);
866        0 :                  le16_add_cpu(&neh->eh_entries, m);
867          :          }
868          :
869          :          set_buffer_uptodate(bh);
870      142 :          unlock_buffer(bh);
871          :
872      142 :          err = ext4_handle_dirty_metadata(handle, inode, bh);
873      142 :          if (err)
874        0 :                  goto cleanup;
875          :          brelse(bh);
876      142 :          bh = NULL;
877          :
878          :          /* correct old leaf */
879      142 :          if (m) {
880        0 :                  err = ext4_ext_get_access(handle, inode, path + depth);
881        0 :                  if (err)
882        0 :                          goto cleanup;
883        0 :                  le16_add_cpu(&path[depth].p_hdr->eh_entries, -m);
884        0 :                  err = ext4_ext_dirty(handle, inode, path + depth);
885        0 :                  if (err)
886        0 :                          goto cleanup;
887          :
888          :          }
889          :
890          :          /* create intermediate indexes */
891      142 :          k = depth - at - 1;
892      142 :          BUG_ON(k < 0);
893          :          if (k)
894          :                  ext_debug("create %d intermediate indices\n", k);
895          :          /* insert new index into current index block */
896          :          /* current depth stored in i var */
897      142 :          i = depth - 1;
898      284 :          while (k--) {
899        0 :                  oldblock = newblock;
900        0 :                  newblock = ablocks[--a];
901        0 :                  bh = sb_getblk(inode->i_sb, newblock);
902        0 :                  if (!bh) {
903        0 :                          err = -EIO;
904        0 :                          goto cleanup;
905          :                  }
906        0 :                  lock_buffer(bh);
907          :
908        0 :                  err = ext4_journal_get_create_access(handle, bh);
909        0 :                  if (err)
910        0 :                          goto cleanup;
```

```
911                :
912              0 :                    neh = ext_block_hdr(bh);
913              0 :                    neh->eh_entries = cpu_to_le16(1);
914              0 :                    neh->eh_magic = EXT4_EXT_MAGIC;
915              0 :                    neh->eh_max = cpu_to_le16(ext4_ext_space_block_idx(inode));
916              0 :                    neh->eh_depth = cpu_to_le16(depth - i);
917              0 :                    fidx = EXT_FIRST_INDEX(neh);
918              0 :                    fidx->ei_block = border;
919                :                    ext4_idx_store_pblock(fidx, oldblock);
920                :
921                :                    ext_debug("int.index at %d (block %llu): %u -> %llu\n",
922                :                              i, newblock, le32_to_cpu(border), oldblock);
923                :                    /* copy indexes */
924              0 :                    m = 0;
925              0 :                    path[i].p_idx++;
926                :
927                :                    ext_debug("cur 0x%p, last 0x%p\n", path[i].p_idx,
928                :                              EXT_MAX_INDEX(path[i].p_hdr));
929              0 :                    BUG_ON(EXT_MAX_INDEX(path[i].p_hdr) !=
930                :                              EXT_LAST_INDEX(path[i].p_hdr));
931              0 :                    while (path[i].p_idx <= EXT_MAX_INDEX(path[i].p_hdr)) {
932                :                            ext_debug("%d: move %d:%llu in new index %llu\n", i,
933                :                                      le32_to_cpu(path[i].p_idx->ei_block),
934                :                                      idx_pblock(path[i].p_idx),
935                :                                      newblock);
936                :                            /*memmove(++fidx, path[i].p_idx++,
937                :                                      sizeof(struct ext4_extent_idx));
938                :                            neh->eh_entries++;
939                :                            BUG_ON(neh->eh_entries > neh->eh_max);*/
940              0 :                            path[i].p_idx++;
941              0 :                            m++;
942                :                    }
943              0 :                    if (m) {
944              0 :                            memmove(++fidx, path[i].p_idx - m,
945                :                                    sizeof(struct ext4_extent_idx) * m);
946              0 :                            le16_add_cpu(&neh->eh_entries, m);
947                :                    }
948                :                    set_buffer_uptodate(bh);
949              0 :                    unlock_buffer(bh);
950                :
951              0 :                    err = ext4_handle_dirty_metadata(handle, inode, bh);
952              0 :                    if (err)
953              0 :                            goto cleanup;
954                :                    brelse(bh);
955              0 :                    bh = NULL;
956                :
957                :                    /* correct old index */
958              0 :                    if (m) {
959              0 :                            err = ext4_ext_get_access(handle, inode, path + i);
960              0 :                            if (err)
961              0 :                                    goto cleanup;
962              0 :                            le16_add_cpu(&path[i].p_hdr->eh_entries, -m);
963              0 :                            err = ext4_ext_dirty(handle, inode, path + i);
964              0 :                            if (err)
965              0 :                                    goto cleanup;
966                :                    }
967                :
968              0 :                    i--;
969                :            }
970                :
971                :            /* insert new index */
972            142 :            err = ext4_ext_insert_index(handle, inode, path + at,
973                :                                        le32_to_cpu(border), newblock);
974                :
975            142 : cleanup:
976            142 :            if (bh) {
977              0 :                    if (buffer_locked(bh))
978              0 :                            unlock_buffer(bh);
979                :                    brelse(bh);
980                :            }
981                :
```

```
    982          142 :          if (err) {
    983              :                  /* free all allocated blocks in error case */
    984            0 :                  for (i = 0; i < depth; i++) {
    985            0 :                          if (!ablocks[i])
    986            0 :                                  continue;
    987            0 :                          ext4_free_blocks(handle, inode, ablocks[i], 1, 1);
    988              :                  }
    989              :          }
    990          142 :          kfree(ablocks);
    991              :
    992          142 :          return err;
    993              : }
    994              :
    995              : /*
    996              :  * ext4_ext_grow_indepth:
    997              :  * implements tree growing procedure:
    998              :  * - allocates new block
    999              :  * - moves top-level data (index block or leaf) into the new block
   1000              :  * - initializes new top-level, creating index that points to the
   1001              :  *   just created block
   1002              :  */
   1003              : static int ext4_ext_grow_indepth(handle_t *handle, struct inode *inode,
   1004              :                                  struct ext4_ext_path *path,
   1005              :                                  struct ext4_extent *newext)
   1006       962359 : {
   1007       962359 :          struct ext4_ext_path *curp = path;
   1008              :          struct ext4_extent_header *neh;
   1009              :          struct ext4_extent_idx *fidx;
   1010              :          struct buffer_head *bh;
   1011              :          ext4_fsblk_t newblock;
   1012       962359 :          int err = 0;
   1013              :
   1014       963197 :          newblock = ext4_ext_new_meta_block(handle, inode, path, newext, &err);
   1015       963197 :          if (newblock == 0)
   1016            0 :                  return err;
   1017              :
   1018      1926247 :          bh = sb_getblk(inode->i_sb, newblock);
   1019       963050 :          if (!bh) {
   1020            0 :                  err = -EIO;
   1021            0 :                  ext4_std_error(inode->i_sb, err);
   1022            0 :                  return err;
   1023              :          }
   1024       963050 :          lock_buffer(bh);
   1025              :
   1026       963941 :          err = ext4_journal_get_create_access(handle, bh);
   1027       962700 :          if (err) {
   1028            0 :                  unlock_buffer(bh);
   1029            0 :                  goto out;
   1030              :          }
   1031              :
   1032              :          /* move top-level index/leaf into new block */
   1033       962700 :          memmove(bh->b_data, curp->p_hdr, sizeof(EXT4_I(inode)->i_data));
   1034              :
   1035              :          /* set size of new block */
   1036       962949 :          neh = ext_block_hdr(bh);
   1037              :          /* old root could have indexes or leaves
   1038              :           * so calculate e_max right way */
   1039       962949 :          if (ext_depth(inode))
   1040            3 :            neh->eh_max = cpu_to_le16(ext4_ext_space_block_idx(inode));
   1041              :          else
   1042       962946 :            neh->eh_max = cpu_to_le16(ext4_ext_space_block(inode));
   1043       962949 :          neh->eh_magic = EXT4_EXT_MAGIC;
   1044              :          set_buffer_uptodate(bh);
   1045       963911 :          unlock_buffer(bh);
   1046              :
   1047       962613 :          err = ext4_handle_dirty_metadata(handle, inode, bh);
   1048       963980 :          if (err)
   1049            0 :                  goto out;
   1050              :
   1051              :          /* create index in new top-level index: num,max,pointer */
   1052       963979 :          err = ext4_ext_get_access(handle, inode, curp);
```

```
1053          963979 :          if (err)
1054               0 :                  goto out;
1055                 :
1056          963979 :          curp->p_hdr->eh_magic = EXT4_EXT_MAGIC;
1057         1927958 :          curp->p_hdr->eh_max = cpu_to_le16(ext4_ext_space_root_idx(inode));
1058          963979 :          curp->p_hdr->eh_entries = cpu_to_le16(1);
1059          963979 :          curp->p_idx = EXT_FIRST_INDEX(curp->p_hdr);
1060                 :
1061          963979 :          if (path[0].p_hdr->eh_depth)
1062               3 :                  curp->p_idx->ei_block =
1063                 :                          EXT_FIRST_INDEX(path[0].p_hdr)->ei_block;
1064                 :          else
1065          963976 :                  curp->p_idx->ei_block =
1066                 :                          EXT_FIRST_EXTENT(path[0].p_hdr)->ee_block;
1067          963979 :          ext4_idx_store_pblock(curp->p_idx, newblock);
1068                 :
1069          963979 :          neh = ext_inode_hdr(inode);
1070          963979 :          fidx = EXT_FIRST_INDEX(neh);
1071                 :          ext_debug("new root: num %d(%d), lblock %d, ptr %llu\n",
1072                 :                    le16_to_cpu(neh->eh_entries), le16_to_cpu(neh->eh_max),
1073                 :                    le32_to_cpu(fidx->ei_block), idx_pblock(fidx));
1074                 :
1075          963979 :          neh->eh_depth = cpu_to_le16(path->p_depth + 1);
1076          963979 :          err = ext4_ext_dirty(handle, inode, curp);
1077          963382 : out:
1078                 :          brelse(bh);
1079                 :
1080          963374 :          return err;
1081                 : }
1082                 :
1083                 : /*
1084                 :  * ext4_ext_create_new_leaf:
1085                 :  * finds empty index and adds new leaf.
1086                 :  * if no free index is found, then it requests in-depth growing.
1087                 :  */
1088                 : static int ext4_ext_create_new_leaf(handle_t *handle, struct inode *inode,
1089                 :                                     struct ext4_ext_path *path,
1090                 :                                     struct ext4_extent *newext)
1091          962559 : {
1092                 :          struct ext4_ext_path *curp;
1093          962559 :          int depth, i, err = 0;
1094                 :
1095          962562 : repeat:
1096          962562 :          i = depth = ext_depth(inode);
1097                 :
1098                 :          /* walk up to the tree and look for free index entry */
1099          962562 :          curp = path + depth;
1100         1925269 :          while (i > 0 && !EXT_HAS_FREE_INDEX(curp)) {
1101             145 :                  i--;
1102             145 :                  curp--;
1103                 :          }
1104                 :
1105                 :          /* we use already allocated block for index block,
1106                 :           * so subsequent data blocks should be contiguous */
1107          962562 :          if (EXT_HAS_FREE_INDEX(curp)) {
1108                 :                  /* if we found index with free entry, then use that
1109                 :                   * entry: create all needed subtree and add new leaf */
1110             142 :                  err = ext4_ext_split(handle, inode, path, newext, i);
1111             142 :                  if (err)
1112               0 :                          goto out;
1113                 :
1114                 :                  /* refill path */
1115             142 :                  ext4_ext_drop_refs(path);
1116             142 :                  path = ext4_ext_find_extent(inode,
1117                 :                                      (ext4_lblk_t)le32_to_cpu(newext->ee_block),
1118                 :                                      path);
1119             142 :                  if (IS_ERR(path))
1120               0 :                          err = PTR_ERR(path);
1121                 :          } else {
1122                 :                  /* tree is full, time to grow in depth */
1123          962420 :                  err = ext4_ext_grow_indepth(handle, inode, path, newext);
```

```
1124          962368 :                 if (err)
1125               0 :                         goto out;
1126                 :
1127                 :                 /* refill path */
1128          962368 :                 ext4_ext_drop_refs(path);
1129          963441 :                 path = ext4_ext_find_extent(inode,
1130                 :                                             (ext4_lblk_t)le32_to_cpu(newext->ee_block),
1131                 :                                             path);
1132          963887 :                 if (IS_ERR(path)) {
1133               0 :                         err = PTR_ERR(path);
1134               0 :                         goto out;
1135                 :                 }
1136                 :
1137                 :                 /*
1138                 :                  * only first (depth 0 -> 1) produces free space;
1139                 :                  * in all other cases we have to split the grown tree
1140                 :                  */
1141          963887 :                 depth = ext_depth(inode);
1142          963887 :                 if (path[depth].p_hdr->eh_entries == path[depth].p_hdr->eh_max) {
1143                 :                         /* now we need to split */
1144               3 :                         goto repeat;
1145                 :                 }
1146                 :         }
1147                 :
1148          964026 : out:
1149          964026 :         return err;
1150                 : }
1151                 :
1152                 : /*
1153                 :  * search the closest allocated block to the left for *logical
1154                 :  * and returns it at @logical + it's physical address at @phys
1155                 :  * if *logical is the smallest allocated block, the function
1156                 :  * returns 0 at @phys
1157                 :  * return value contains 0 (success) or error code
1158                 :  */
1159                 : int
1160                 : ext4_ext_search_left(struct inode *inode, struct ext4_ext_path *path,
1161                 :                     ext4_lblk_t *logical, ext4_fsblk_t *phys)
1162       220060745 : {
1163                 :         struct ext4_extent_idx *ix;
1164                 :         struct ext4_extent *ex;
1165                 :         int depth, ee_len;
1166                 :
1167       220060745 :         BUG_ON(path == NULL);
1168       220122765 :         depth = path->p_depth;
1169       220122765 :         *phys = 0;
1170                 :
1171       220122765 :         if (depth == 0 && path->p_ext == NULL)
1172         9211048 :                 return 0;
1173                 :
1174                 :         /* usually extent in the path covers blocks smaller
1175                 :          * then *logical, but it can be that extent is the
1176                 :          * first one in the file */
1177                 :
1178       210911717 :         ex = path[depth].p_ext;
1179       210911717 :         ee_len = ext4_ext_get_actual_len(ex);
1180       210911717 :         if (*logical < le32_to_cpu(ex->ee_block)) {
1181               9 :                 BUG_ON(EXT_FIRST_EXTENT(path[depth].p_hdr) != ex);
1182              13 :                 while (--depth >= 0) {
1183               4 :                         ix = path[depth].p_idx;
1184               4 :                         BUG_ON(ix != EXT_FIRST_INDEX(path[depth].p_hdr));
1185                 :                 }
1186               9 :                 return 0;
1187                 :         }
1188                 :
1189       210911708 :         BUG_ON(*logical < (le32_to_cpu(ex->ee_block) + ee_len));
1190                 :
1191       210697939 :         *logical = le32_to_cpu(ex->ee_block) + ee_len - 1;
1192       210697939 :         *phys = ext_pblock(ex) + ee_len - 1;
1193       210697939 :         return 0;
1194                 : }
```

```
                :
                : /*
                :  * search the closest allocated block to the right for *logical
                :  * and returns it at @logical + it's physical address at @phys
                :  * if *logical is the smallest allocated block, the function
                :  * returns 0 at @phys
                :  * return value contains 0 (success) or error code
                :  */
                : int
                : ext4_ext_search_right(struct inode *inode, struct ext4_ext_path *path,
                :                       ext4_lblk_t *logical, ext4_fsblk_t *phys)
      219733027 : {
      219733027 :         struct buffer_head *bh = NULL;
                :         struct ext4_extent_header *eh;
                :         struct ext4_extent_idx *ix;
                :         struct ext4_extent *ex;
                :         ext4_fsblk_t block;
                :         int depth;      /* Note, NOT eh_depth; depth from top of tree */
                :         int ee_len;
                :
      219733027 :         BUG_ON(path == NULL);
      219723422 :         depth = path->p_depth;
      219723422 :         *phys = 0;
                :
      219723422 :         if (depth == 0 && path->p_ext == NULL)
        9211034 :                 return 0;
                :
                :         /* usually extent in the path covers blocks smaller
                :          * then *logical, but it can be that extent is the
                :          * first one in the file */
                :
      210512388 :         ex = path[depth].p_ext;
      210512388 :         ee_len = ext4_ext_get_actual_len(ex);
      210512388 :         if (*logical < le32_to_cpu(ex->ee_block)) {
              9 :                 BUG_ON(EXT_FIRST_EXTENT(path[depth].p_hdr) != ex);
             13 :                 while (--depth >= 0) {
              4 :                         ix = path[depth].p_idx;
              4 :                         BUG_ON(ix != EXT_FIRST_INDEX(path[depth].p_hdr));
                :                 }
              9 :                 *logical = le32_to_cpu(ex->ee_block);
              9 :                 *phys = ext_pblock(ex);
              9 :                 return 0;
                :         }
                :
      210512379 :         BUG_ON(*logical < (le32_to_cpu(ex->ee_block) + ee_len));
                :
      210718298 :         if (ex != EXT_LAST_EXTENT(path[depth].p_hdr)) {
                :                 /* next allocated block in this leaf */
          34527 :                 ex++;
          34527 :                 *logical = le32_to_cpu(ex->ee_block);
          34527 :                 *phys = ext_pblock(ex);
          34527 :                 return 0;
                :         }
                :
                :         /* go up and search for index to the right */
      467818229 :         while (--depth >= 0) {
      257134458 :                 ix = path[depth].p_idx;
      257134458 :                 if (ix != EXT_LAST_INDEX(path[depth].p_hdr))
              0 :                         goto got_index;
                :         }
                :
                :         /* we've gone up to the root and found no index to the right */
      210683771 :         return 0;
                :
              0 : got_index:
                :         /* we've found index to the right, let's
                :          * follow it and find the closest allocated
                :          * block to the right */
              0 :         ix++;
              0 :         block = idx_pblock(ix);
              0 :         while (++depth < path->p_depth) {
```

```
1266                0 :                  bh = sb_bread(inode->i_sb, block);
1267                0 :                  if (bh == NULL)
1268                0 :                          return -EIO;
1269                0 :                  eh = ext_block_hdr(bh);
1270                  :                  /* subtract from p_depth to get proper eh_depth */
1271                0 :                  if (ext4_ext_check(inode, eh, path->p_depth - depth)) {
1272                  :                          put_bh(bh);
1273                0 :                          return -EIO;
1274                  :                  }
1275                0 :                  ix = EXT_FIRST_INDEX(eh);
1276                0 :                  block = idx_pblock(ix);
1277                  :                  put_bh(bh);
1278                  :          }
1279                  :
1280                0 :          bh = sb_bread(inode->i_sb, block);
1281                0 :          if (bh == NULL)
1282                0 :                  return -EIO;
1283                0 :          eh = ext_block_hdr(bh);
1284                0 :          if (ext4_ext_check(inode, eh, path->p_depth - depth)) {
1285                  :                  put_bh(bh);
1286                0 :                  return -EIO;
1287                  :          }
1288                0 :          ex = EXT_FIRST_EXTENT(eh);
1289                0 :          *logical = le32_to_cpu(ex->ee_block);
1290                0 :          *phys = ext_pblock(ex);
1291                  :          put_bh(bh);
1292                0 :          return 0;
1293                  : }
1294                  :
1295                  : /*
1296                  :  * ext4_ext_next_allocated_block:
1297                  :  * returns allocated block in subsequent extent or EXT_MAX_BLOCK.
1298                  :  * NOTE: it considers block number from index entry as
1299                  :  * allocated block. Thus, index entries have to be consistent
1300                  :  * with leaves.
1301                  :  */
1302                  : static ext4_lblk_t
1303                  : ext4_ext_next_allocated_block(struct ext4_ext_path *path)
1304        422245335 : {
1305                  :          int depth;
1306                  :
1307        422245335 :          BUG_ON(path == NULL);
1308        423334218 :          depth = path->p_depth;
1309                  :
1310        423334218 :          if (depth == 0 && path->p_ext == NULL)
1311                0 :                  return EXT_MAX_BLOCK;
1312                  :
1313       1363045351 :          while (depth >= 0) {
1314        938707666 :                  if (depth == path->p_depth) {
1315                  :                          /* leaf */
1316        422133283 :                          if (path[depth].p_ext !=
1317                  :                                          EXT_LAST_EXTENT(path[depth].p_hdr))
1318                0 :                            return le32_to_cpu(path[depth].p_ext[1].ee_block);
1319                  :                  } else {
1320                  :                          /* index */
1321        516574383 :                          if (path[depth].p_idx !=
1322                  :                                          EXT_LAST_INDEX(path[depth].p_hdr))
1323                0 :                            return le32_to_cpu(path[depth].p_idx[1].ei_block);
1324                  :                  }
1325        939711133 :                  depth--;
1326                  :          }
1327                  :
1328        424337685 :          return EXT_MAX_BLOCK;
1329                  : }
1330                  :
1331                  : /*
1332                  :  * ext4_ext_next_leaf_block:
1333                  :  * returns first allocated block from next leaf or EXT_MAX_BLOCK
1334                  :  */
1335                  : static ext4_lblk_t ext4_ext_next_leaf_block(struct inode *inode,
1336                  :                                             struct ext4_ext_path *path)
```

```
1337                :  {
1338                :          int depth;
1339                :
1340          962570 :          BUG_ON(path == NULL);
1341          962526 :          depth = path->p_depth;
1342                :
1343                :          /* zero-tree has no leaf blocks at all */
1344          962526 :          if (depth == 0)
1345          962384 :                  return EXT_MAX_BLOCK;
1346                :
1347                :          /* go to index block */
1348             142 :          depth--;
1349                :
1350             319 :          while (depth >= 0) {
1351             177 :                  if (path[depth].p_idx !=
1352                :                                  EXT_LAST_INDEX(path[depth].p_hdr))
1353               0 :                          return (ext4_lblk_t)
1354                :                                  le32_to_cpu(path[depth].p_idx[1].ei_block);
1355             177 :                  depth--;
1356                :          }
1357                :
1358             142 :          return EXT_MAX_BLOCK;
1359                :  }
1360                :
1361                :  /*
1362                :   * ext4_ext_correct_indexes:
1363                :   * if leaf gets modified and modified extent is first in the leaf,
1364                :   * then we have to correct all indexes above.
1365                :   * TODO: do we need to correct tree in all cases?
1366                :   */
1367                :  static int ext4_ext_correct_indexes(handle_t *handle, struct inode *inode,
1368                :                                      struct ext4_ext_path *path)
1369     219661013 :  {
1370                :          struct ext4_extent_header *eh;
1371     219661013 :          int depth = ext_depth(inode);
1372                :          struct ext4_extent *ex;
1373                :          __le32 border;
1374     219661013 :          int k, err = 0;
1375                :
1376     219661013 :          eh = path[depth].p_hdr;
1377     219661013 :          ex = path[depth].p_ext;
1378     219661013 :          BUG_ON(ex == NULL);
1379     219855404 :          BUG_ON(eh == NULL);
1380                :
1381     219889635 :          if (depth == 0) {
1382                :                  /* there is no tree at all */
1383      46431056 :                  return 0;
1384                :          }
1385                :
1386     173458579 :          if (ex != EXT_FIRST_EXTENT(eh)) {
1387                :                  /* we correct tree if first leaf got modified only */
1388     173405073 :                  return 0;
1389                :          }
1390                :
1391                :          /*
1392                :           * TODO: we need correction if border is smaller than current one
1393                :           */
1394          53506 :          k = depth - 1;
1395          53506 :          border = path[depth].p_ext->ee_block;
1396         107012 :          err = ext4_ext_get_access(handle, inode, path + k);
1397          53506 :          if (err)
1398              0 :                  return err;
1399          53506 :          path[k].p_idx->ei_block = border;
1400          53506 :          err = ext4_ext_dirty(handle, inode, path + k);
1401          53506 :          if (err)
1402              0 :                  return err;
1403                :
1404          53506 :          while (k--) {
1405                :                  /* change all left-side indexes */
1406          47230 :                  if (path[k+1].p_idx != EXT_FIRST_INDEX(path[k+1].p_hdr))
1407          47230 :                          break;
```

```
1408              0 :                         err = ext4_ext_get_access(handle, inode, path + k);
1409              0 :                         if (err)
1410              0 :                                 break;
1411              0 :                         path[k].p_idx->ei_block = border;
1412              0 :                         err = ext4_ext_dirty(handle, inode, path + k);
1413              0 :                         if (err)
1414              0 :                                 break;
1415                :                 }
1416                :
1417          53506 :                 return err;
1418                : }
1419                :
1420                : int
1421                : ext4_can_extents_be_merged(struct inode *inode, struct ext4_extent *ex1,
1422                :                            struct ext4_extent *ex2)
1423      209233701 : {
1424                :         unsigned short ext1_ee_len, ext2_ee_len, max_len;
1425                :
1426                :         /*
1427                :          * Make sure that either both extents are uninitialized, or
1428                :          * both are _not_.
1429                :          */
1430      209233701 :         if (ext4_ext_is_uninitialized(ex1) ^ ext4_ext_is_uninitialized(ex2))
1431             18 :                 return 0;
1432                :
1433      209233683 :         if (ext4_ext_is_uninitialized(ex1))
1434              0 :                 max_len = EXT_UNINIT_MAX_LEN;
1435                :         else
1436      209405124 :                 max_len = EXT_INIT_MAX_LEN;
1437                :
1438      209233683 :         ext1_ee_len = ext4_ext_get_actual_len(ex1);
1439      209233683 :         ext2_ee_len = ext4_ext_get_actual_len(ex2);
1440                :
1441      209233683 :         if (le32_to_cpu(ex1->ee_block) + ext1_ee_len !=
1442                :                         le32_to_cpu(ex2->ee_block))
1443          47285 :                 return 0;
1444                :
1445                :         /*
1446                :          * To allow future support for preallocated extents to be added
1447                :          * as an RO_COMPAT feature, refuse to merge to extents if
1448                :          * this can result in the top bit of ee_len being set.
1449                :          */
1450      209186398 :         if (ext1_ee_len + ext2_ee_len > max_len)
1451          14037 :                 return 0;
1452                : #ifdef AGGRESSIVE_TEST
1453                :         if (ext1_ee_len >= 4)
1454                :                 return 0;
1455                : #endif
1456                :
1457      418344722 :         if (ext_pblock(ex1) + ext1_ee_len == ext_pblock(ex2))
1458      198594112 :                 return 1;
1459       10578249 :         return 0;
1460                : }
1461                :
1462                : /*
1463                :  * This function tries to merge the "ex" extent to the next extent in the tree.
1464                :  * It always tries to merge towards right. If you want to merge towards
1465                :  * left, pass "ex - 1" as argument instead of "ex".
1466                :  * Returns 0 if the extents (ex and ex+1) were _not_ merged and returns
1467                :  * 1 if they got merged.
1468                :  */
1469                : int ext4_ext_try_to_merge(struct inode *inode,
1470                :                           struct ext4_ext_path *path,
1471                :                           struct ext4_extent *ex)
1472      219762677 : {
1473                :         struct ext4_extent_header *eh;
1474                :         unsigned int depth, len;
1475      219762677 :         int merge_done = 0;
1476      219762677 :         int uninitialized = 0;
1477                :
1478      219762677 :         depth = ext_depth(inode);
```

```
1479      219762677 :          BUG_ON(path[depth].p_hdr == NULL);
1480      219744636 :          eh = path[depth].p_hdr;
1481                :
1482      439525000 :          while (ex < EXT_LAST_EXTENT(eh)) {
1483          48922 :                  if (!ext4_can_extents_be_merged(inode, ex, ex + 1))
1484          20097 :                          break;
1485                :                  /* merge with next extent! */
1486          28850 :                  if (ext4_ext_is_uninitialized(ex))
1487              0 :                          uninitialized = 1;
1488          57700 :                  ex->ee_len = cpu_to_le16(ext4_ext_get_actual_len(ex)
1489                :                                   + ext4_ext_get_actual_len(ex + 1));
1490          28850 :                  if (uninitialized)
1491                :                          ext4_ext_mark_uninitialized(ex);
1492                :
1493          28850 :                  if (ex + 1 < EXT_LAST_EXTENT(eh)) {
1494          14375 :                          len = (EXT_LAST_EXTENT(eh) - ex - 1)
1495                :                                  * sizeof(struct ext4_extent);
1496          14375 :                          memmove(ex + 1, ex + 2, len);
1497                :                  }
1498          28861 :                  le16_add_cpu(&eh->eh_entries, -1);
1499          28861 :                  merge_done = 1;
1500          28861 :                  WARN_ON(eh->eh_entries == 0);
1501          28853 :                  if (!eh->eh_entries)
1502              0 :                          ext4_error(inode->i_sb, "ext4_ext_try_to_merge",
1503                :                             "inode#%lu, eh->eh_entries = 0!", inode->i_ino);
1504                :          }
1505                :
1506      219751539 :          return merge_done;
1507                : }
1508                :
1509                : /*
1510                :  * check if a portion of the "newext" extent overlaps with an
1511                :  * existing extent.
1512                :  *
1513                :  * If there is an overlap discovered, it updates the length of the newext
1514                :  * such that there will be no overlap, and then returns 1.
1515                :  * If there is no overlap found, it returns 0.
1516                :  */
1517                : unsigned int ext4_ext_check_overlap(struct inode *inode,
1518                :                                     struct ext4_extent *newext,
1519                :                                     struct ext4_ext_path *path)
1520      220769649 : {
1521                :          ext4_lblk_t b1, b2;
1522                :          unsigned int depth, len1;
1523      220769649 :          unsigned int ret = 0;
1524                :
1525      220769649 :          b1 = le32_to_cpu(newext->ee_block);
1526      220769649 :          len1 = ext4_ext_get_actual_len(newext);
1527      220769649 :          depth = ext_depth(inode);
1528      220769649 :          if (!path[depth].p_ext)
1529        9211026 :                  goto out;
1530      211558623 :          b2 = le32_to_cpu(path[depth].p_ext->ee_block);
1531                :
1532                :          /*
1533                :           * get the next allocated block if the extent in the path
1534                :           * is before the requested block(s)
1535                :           */
1536      211558623 :          if (b2 < b1) {
1537      211729358 :                  b2 = ext4_ext_next_allocated_block(path);
1538      212300075 :                  if (b2 == EXT_MAX_BLOCK)
1539      212222709 :                          goto out;
1540                :          }
1541                :
1542                :          /* check for wrap through zero on extent logical start block*/
1543              0 :          if (b1 + len1 < b1) {
1544              0 :                  len1 = EXT_MAX_BLOCK - b1;
1545              0 :                  newext->ee_len = cpu_to_le16(len1);
1546              0 :                  ret = 1;
1547                :          }
1548                :
1549                :          /* check for overlap */
```

```
1550                0 :         if (b1 + len1 > b2) {
1551                4 :                 newext->ee_len = cpu_to_le16(b2 - b1);
1552                4 :                 ret = 1;
1553                  :         }
1554        221340366 : out:
1555        221340366 :         return ret;
1556                  : }
1557                  :
1558                  : /*
1559                  :  * ext4_ext_insert_extent:
1560                  :  * tries to merge requsted extent into the existing extent or
1561                  :  * inserts requested extent as new one into the tree,
1562                  :  * creating new leaf in the no-space case.
1563                  :  */
1564                  : int ext4_ext_insert_extent(handle_t *handle, struct inode *inode,
1565                  :                                 struct ext4_ext_path *path,
1566                  :                                 struct ext4_extent *newext)
1567        218251911 : {
1568                  :         struct ext4_extent_header *eh;
1569                  :         struct ext4_extent *ex, *fex;
1570                  :         struct ext4_extent *nearex; /* nearest extent */
1571        218251911 :         struct ext4_ext_path *npath = NULL;
1572                  :         int depth, len, err;
1573                  :         ext4_lblk_t next;
1574        218251911 :         unsigned uninitialized = 0;
1575                  :
1576        218251911 :         BUG_ON(ext4_ext_get_actual_len(newext) == 0);
1577        218624516 :         depth = ext_depth(inode);
1578        218624516 :         ex = path[depth].p_ext;
1579        218624516 :         BUG_ON(path[depth].p_hdr == NULL);
1580                  :
1581                  :         /* try to insert block into found extent and return */
1582        220067095 :         if (ex && ext4_can_extents_be_merged(inode, ex, newext)) {
1583                  :                 ext_debug("append %d block to %d:%d (from %llu)\n",
1584                  :                                 ext4_ext_get_actual_len(newext),
1585                  :                                 le32_to_cpu(ex->ee_block),
1586                  :                                 ext4_ext_get_actual_len(ex), ext_pblock(ex));
1587        400604099 :                 err = ext4_ext_get_access(handle, inode, path + depth);
1588        199974918 :                 if (err)
1589                0 :                         return err;
1590                  :
1591                  :                 /*
1592                  :                  * ext4_can_extents_be_merged should have checked that either
1593                  :                  * both extents are uninitialized, or both aren't. Thus we
1594                  :                  * need to check only one of them here.
1595                  :                  */
1596        199974918 :                 if (ext4_ext_is_uninitialized(ex))
1597                2 :                         uninitialized = 1;
1598        399949836 :                 ex->ee_len = cpu_to_le16(ext4_ext_get_actual_len(ex)
1599                  :                                          + ext4_ext_get_actual_len(newext));
1600        199974918 :                 if (uninitialized)
1601                  :                         ext4_ext_mark_uninitialized(ex);
1602        199974918 :                 eh = path[depth].p_hdr;
1603        199974918 :                 nearex = ex;
1604        199974918 :                 goto merge;
1605                  :         }
1606                  :
1607         19833523 : repeat:
1608         19833523 :         depth = ext_depth(inode);
1609         19833523 :         eh = path[depth].p_hdr;
1610         19833523 :         if (le16_to_cpu(eh->eh_entries) < le16_to_cpu(eh->eh_max))
1611         18870953 :                 goto has_space;
1612                  :
1613                  :         /* probably next leaf has space for us? */
1614           962570 :         fex = EXT_LAST_EXTENT(eh);
1615           962526 :         next = ext4_ext_next_leaf_block(inode, path);
1616           962526 :         if (le32_to_cpu(newext->ee_block) > le32_to_cpu(fex->ee_block)
1617                  :             && next != EXT_MAX_BLOCK) {
1618                  :                 ext_debug("next leaf block - %d\n", next);
1619                0 :                 BUG_ON(npath != NULL);
1620                0 :                 npath = ext4_ext_find_extent(inode, next, NULL);
```

```
    1621                0 :                     if (IS_ERR(npath))
    1622                0 :                             return PTR_ERR(npath);
    1623                0 :                     BUG_ON(npath->p_depth != path->p_depth);
    1624                0 :                     eh = npath[depth].p_hdr;
    1625                0 :                     if (le16_to_cpu(eh->eh_entries) < le16_to_cpu(eh->eh_max)) {
    1626                  :                             ext_debug("next leaf isnt full(%d)\n",
    1627                  :                                       le16_to_cpu(eh->eh_entries));
    1628                0 :                             path = npath;
    1629                0 :                             goto repeat;
    1630                  :                     }
    1631                  :                     ext_debug("next leaf has no free space(%d,%d)\n",
    1632                  :                               le16_to_cpu(eh->eh_entries), le16_to_cpu(eh->eh_max));
    1633                  :             }
    1634                  :
    1635                  :             /*
    1636                  :              * There is no free space in the found leaf.
    1637                  :              * We're gonna add a new leaf in the tree.
    1638                  :              */
    1639           962526 :             err = ext4_ext_create_new_leaf(handle, inode, path, newext);
    1640           962244 :             if (err)
    1641                0 :                     goto cleanup;
    1642           962244 :             depth = ext_depth(inode);
    1643           962244 :             eh = path[depth].p_hdr;
    1644                  :
    1645         19833197 : has_space:
    1646         19833197 :             nearex = path[depth].p_ext;
    1647                  :
    1648         39671384 :             err = ext4_ext_get_access(handle, inode, path + depth);
    1649         19838187 :             if (err)
    1650                0 :                     goto cleanup;
    1651                  :
    1652         19838187 :             if (!nearex) {
    1653                  :                     /* there is no extent in this leaf, create first one */
    1654                  :                     ext_debug("first extent in the leaf: %d:%llu:%d\n",
    1655                  :                                     le32_to_cpu(newext->ee_block),
    1656                  :                                     ext_pblock(newext),
    1657                  :                                     ext4_ext_get_actual_len(newext));
    1658          9196890 :                     path[depth].p_ext = EXT_FIRST_EXTENT(eh);
    1659         10641297 :             } else if (le32_to_cpu(newext->ee_block)
    1660                  :                            > le32_to_cpu(nearex->ee_block)) {
    1661                  : /*                  BUG_ON(newext->ee_block == nearex->ee_block); */
    1662         10641288 :                     if (nearex != EXT_LAST_EXTENT(eh)) {
    1663            19477 :                             len = EXT_MAX_EXTENT(eh) - nearex;
    1664            19477 :                             len = (len - 1) * sizeof(struct ext4_extent);
    1665            19477 :                             len = len < 0 ? 0 : len;
    1666                  :                             ext_debug("insert %d:%llu:%d after: nearest 0x%p, "
    1667                  :                                             "move %d from 0x%p to 0x%p\n",
    1668                  :                                             le32_to_cpu(newext->ee_block),
    1669                  :                                             ext_pblock(newext),
    1670                  :                                             ext4_ext_get_actual_len(newext),
    1671                  :                                             nearex, len, nearex + 1, nearex + 2);
    1672            19477 :                             memmove(nearex + 2, nearex + 1, len);
    1673                  :                     }
    1674         10646748 :                     path[depth].p_ext = nearex + 1;
    1675                  :             } else {
    1676                9 :                     BUG_ON(newext->ee_block == nearex->ee_block);
    1677                9 :                     len = (EXT_MAX_EXTENT(eh) - nearex) * sizeof(struct ext4_extent);
    1678                9 :                     len = len < 0 ? 0 : len;
    1679                  :                     ext_debug("insert %d:%llu:%d before: nearest 0x%p, "
    1680                  :                                     "move %d from 0x%p to 0x%p\n",
    1681                  :                                     le32_to_cpu(newext->ee_block),
    1682                  :                                     ext_pblock(newext),
    1683                  :                                     ext4_ext_get_actual_len(newext),
    1684                  :                                     nearex, len, nearex + 1, nearex + 2);
    1685                9 :                     memmove(nearex + 1, nearex, len);
    1686                9 :                     path[depth].p_ext = nearex;
    1687                  :             }
    1688                  :
    1689         19843647 :             le16_add_cpu(&eh->eh_entries, 1);
    1690         19843647 :             nearex = path[depth].p_ext;
    1691         19843647 :             nearex->ee_block = newext->ee_block;
```

```
1692                :          ext4_ext_store_pblock(nearex, ext_pblock(newext));
1693         19843647 :          nearex->ee_len = newext->ee_len;
1694                :
1695        219818565 : merge:
1696                :          /* try to merge extents to the right */
1697        219818565 :          ext4_ext_try_to_merge(inode, path, nearex);
1698                :
1699                :          /* try to merge extents to the left */
1700                :
1701                :          /* time to correct all indexes above */
1702        219720086 :          err = ext4_ext_correct_indexes(handle, inode, path);
1703        220416071 :          if (err)
1704                0 :                  goto cleanup;
1705                :
1706        220416071 :          err = ext4_ext_dirty(handle, inode, path + depth);
1707                :
1708        219845957 : cleanup:
1709        219845957 :          if (npath) {
1710                0 :                  ext4_ext_drop_refs(npath);
1711                0 :                  kfree(npath);
1712                :          }
1713                :          ext4_ext_invalidate_cache(inode);
1714        220032661 :          return err;
1715                : }
1716                :
1717                : int ext4_ext_walk_space(struct inode *inode, ext4_lblk_t block,
1718                :                         ext4_lblk_t num, ext_prepare_callback func,
1719                :                         void *cbdata)
1720                0 : {
1721                0 :          struct ext4_ext_path *path = NULL;
1722                :          struct ext4_ext_cache cbex;
1723                :          struct ext4_extent *ex;
1724                0 :          ext4_lblk_t next, start = 0, end = 0;
1725                0 :          ext4_lblk_t last = block + num;
1726                0 :          int depth, exists, err = 0;
1727                :
1728                0 :          BUG_ON(func == NULL);
1729                0 :          BUG_ON(inode == NULL);
1730                :
1731                0 :          while (block < last && block != EXT_MAX_BLOCK) {
1732                0 :                  num = last - block;
1733                :                  /* find extent for this block */
1734                0 :                  path = ext4_ext_find_extent(inode, block, path);
1735                0 :                  if (IS_ERR(path)) {
1736                0 :                          err = PTR_ERR(path);
1737                0 :                          path = NULL;
1738                0 :                          break;
1739                :                  }
1740                :
1741                0 :                  depth = ext_depth(inode);
1742                0 :                  BUG_ON(path[depth].p_hdr == NULL);
1743                0 :                  ex = path[depth].p_ext;
1744                0 :                  next = ext4_ext_next_allocated_block(path);
1745                :
1746                0 :                  exists = 0;
1747                0 :                  if (!ex) {
1748                :                          /* there is no extent yet, so try to allocate
1749                :                           * all requested space */
1750                0 :                          start = block;
1751                0 :                          end = block + num;
1752                0 :                  } else if (le32_to_cpu(ex->ee_block) > block) {
1753                :                          /* need to allocate space before found extent */
1754                0 :                          start = block;
1755                0 :                          end = le32_to_cpu(ex->ee_block);
1756                0 :                          if (block + num < end)
1757                0 :                                  end = block + num;
1758                0 :                  } else if (block >= le32_to_cpu(ex->ee_block)
1759                :                                          + ext4_ext_get_actual_len(ex)) {
1760                :                          /* need to allocate space after found extent */
1761                0 :                          start = block;
1762                0 :                          end = block + num;
```

```
1763                0 :                             if (end >= next)
1764                0 :                                     end = next;
1765                0 :                     } else if (block >= le32_to_cpu(ex->ee_block)) {
1766                  :                             /*
1767                  :                              * some part of requested space is covered
1768                  :                              * by found extent
1769                  :                              */
1770                0 :                             start = block;
1771                0 :                             end = le32_to_cpu(ex->ee_block)
1772                  :                                     + ext4_ext_get_actual_len(ex);
1773                0 :                             if (block + num < end)
1774                0 :                                     end = block + num;
1775                0 :                             exists = 1;
1776                  :                     } else {
1777                0 :                             BUG();
1778                  :                     }
1779                0 :                     BUG_ON(end <= start);
1780                  :
1781                0 :                     if (!exists) {
1782                0 :                             cbex.ec_block = start;
1783                0 :                             cbex.ec_len = end - start;
1784                0 :                             cbex.ec_start = 0;
1785                0 :                             cbex.ec_type = EXT4_EXT_CACHE_GAP;
1786                  :                     } else {
1787                0 :                             cbex.ec_block = le32_to_cpu(ex->ee_block);
1788                0 :                             cbex.ec_len = ext4_ext_get_actual_len(ex);
1789                0 :                             cbex.ec_start = ext_pblock(ex);
1790                0 :                             cbex.ec_type = EXT4_EXT_CACHE_EXTENT;
1791                  :                     }
1792                  :
1793                0 :                     BUG_ON(cbex.ec_len == 0);
1794                0 :                     err = func(inode, path, &cbex, ex, cbdata);
1795                0 :                     ext4_ext_drop_refs(path);
1796                  :
1797                0 :                     if (err < 0)
1798                0 :                             break;
1799                  :
1800                0 :                     if (err == EXT_REPEAT)
1801                0 :                             continue;
1802                0 :                     else if (err == EXT_BREAK) {
1803                0 :                             err = 0;
1804                0 :                             break;
1805                  :                     }
1806                  :
1807                0 :                     if (ext_depth(inode) != depth) {
1808                  :                             /* depth was changed. we have to realloc path */
1809                0 :                             kfree(path);
1810                0 :                             path = NULL;
1811                  :                     }
1812                  :
1813                0 :                     block = cbex.ec_block + cbex.ec_len;
1814                  :             }
1815                  :
1816                0 :             if (path) {
1817                0 :                     ext4_ext_drop_refs(path);
1818                0 :                     kfree(path);
1819                  :             }
1820                  :
1821                0 :             return err;
1822                  : }
1823                  :
1824                  : static void
1825                  : ext4_ext_put_in_cache(struct inode *inode, ext4_lblk_t block,
1826                  :                       __u32 len, ext4_fsblk_t start, int type)
1827                  : {
1828                  :             struct ext4_ext_cache *cex;
1829        494665501 :             BUG_ON(len == 0);
1830        493589269 :             spin_lock(&EXT4_I(inode)->i_block_reservation_lock);
1831        493521305 :             cex = &EXT4_I(inode)->i_cached_extent;
1832        493521305 :             cex->ec_type = type;
1833        493521305 :             cex->ec_block = block;
```

```
      493521305 :         cex->ec_len = len;
      493521305 :         cex->ec_start = start;
      493521305 :         spin_unlock(&EXT4_I(inode)->i_block_reservation_lock);
                : }
                :
                : /*
                :  * ext4_ext_put_gap_in_cache:
                :  * calculate boundaries of the gap that the requested block fits into
                :  * and cache this gap
                :  */
                : static void
                : ext4_ext_put_gap_in_cache(struct inode *inode, struct ext4_ext_path *path,
                :                           ext4_lblk_t block)
                : {
      219867001 :         int depth = ext_depth(inode);
                :         unsigned long len;
                :         ext4_lblk_t lblock;
                :         struct ext4_extent *ex;
                :
      219867001 :         ex = path[depth].p_ext;
      219867001 :         if (ex == NULL) {
                :                 /* there is no extent yet, so gap is [0;-] */
        9212494 :                 lblock = 0;
        9212494 :                 len = EXT_MAX_BLOCK;
                :                 ext_debug("cache gap(whole file):");
      210654507 :         } else if (block < le32_to_cpu(ex->ee_block)) {
              9 :                 lblock = block;
              9 :                 len = le32_to_cpu(ex->ee_block) - block;
                :                 ext_debug("cache gap(before): %u [%u:%u]",
                :                           block,
                :                           le32_to_cpu(ex->ee_block),
                :                            ext4_ext_get_actual_len(ex));
      421308996 :         } else if (block >= le32_to_cpu(ex->ee_block)
                :                         + ext4_ext_get_actual_len(ex)) {
                :                 ext4_lblk_t next;
      421308996 :                 lblock = le32_to_cpu(ex->ee_block)
                :                         + ext4_ext_get_actual_len(ex);
                :
      210654498 :                 next = ext4_ext_next_allocated_block(path);
                :                 ext_debug("cache gap(after): [%u:%u] %u",
                :                           le32_to_cpu(ex->ee_block),
                :                           ext4_ext_get_actual_len(ex),
                :                           block);
      212299977 :                 BUG_ON(next == lblock);
      212316881 :                 len = next - lblock;
                :         } else {
              0 :                 lblock = len = 0;
              0 :                 BUG();
                :         }
                :
                :         ext_debug(" -> %u:%lu\n", lblock, len);
                :         ext4_ext_put_in_cache(inode, lblock, len, 0, EXT4_EXT_CACHE_GAP);
                : }
                :
                : static int
                : ext4_ext_in_cache(struct inode *inode, ext4_lblk_t block,
                :                   struct ext4_extent *ex)
                : {
                :         struct ext4_ext_cache *cex;
     1331774930 :         int ret = EXT4_EXT_CACHE_NO;
                :
                :         /*
                :          * We borrow i_block_reservation_lock to protect i_cached_extent
                :          */
     1331774930 :         spin_lock(&EXT4_I(inode)->i_block_reservation_lock);
     1338331602 :         cex = &EXT4_I(inode)->i_cached_extent;
                :
                :         /* has cache valid data? */
     1338331602 :         if (cex->ec_type == EXT4_EXT_CACHE_NO)
                :                 goto errout;
                :
```

```
1905  1331130430 :          BUG_ON(cex->ec_type != EXT4_EXT_CACHE_GAP &&
1906             :                                 cex->ec_type != EXT4_EXT_CACHE_EXTENT);
1907  1313894273 :          if (block >= cex->ec_block && block < cex->ec_block + cex->ec_len) {
1908  1063669108 :                  ex->ee_block = cpu_to_le32(cex->ec_block);
1909  1063669108 :                  ext4_ext_store_pblock(ex, cex->ec_start);
1910  1063669108 :                  ex->ee_len = cpu_to_le16(cex->ec_len);
1911             :                  ext_debug("%u cached by %u:%u:%llu\n",
1912             :                                  block,
1913             :                                  cex->ec_block, cex->ec_len, cex->ec_start);
1914  1063669108 :                  ret = cex->ec_type;
1915             :          }
1916  1321095445 : errout:
1917  1321095445 :          spin_unlock(&EXT4_I(inode)->i_block_reservation_lock);
1918  1348151084 :          return ret;
1919             : }
1920             :
1921             : /*
1922             :  * ext4_ext_rm_idx:
1923             :  * removes index from the index block.
1924             :  * It's used in truncate case only, thus all requests are for
1925             :  * last index in the block only.
1926             :  */
1927             : static int ext4_ext_rm_idx(handle_t *handle, struct inode *inode,
1928             :                            struct ext4_ext_path *path)
1929   215753 : {
1930             :          struct buffer_head *bh;
1931             :          int err;
1932             :          ext4_fsblk_t leaf;
1933             :
1934             :          /* free index block */
1935   215753 :          path--;
1936   431506 :          leaf = idx_pblock(path->p_idx);
1937   215753 :          BUG_ON(path->p_hdr->eh_entries == 0);
1938   215753 :          err = ext4_ext_get_access(handle, inode, path);
1939   215753 :          if (err)
1940        0 :                  return err;
1941   215753 :          le16_add_cpu(&path->p_hdr->eh_entries, -1);
1942   215753 :          err = ext4_ext_dirty(handle, inode, path);
1943   215753 :          if (err)
1944        0 :                  return err;
1945             :          ext_debug("index is empty, remove it, free block %llu\n", leaf);
1946   431506 :          bh = sb_find_get_block(inode->i_sb, leaf);
1947   215753 :          ext4_forget(handle, 1, inode, bh, leaf);
1948   215753 :          ext4_free_blocks(handle, inode, leaf, 1, 1);
1949   215753 :          return err;
1950             : }
1951             :
1952             : /*
1953             :  * ext4_ext_calc_credits_for_single_extent:
1954             :  * This routine returns max. credits that needed to insert an extent
1955             :  * to the extent tree.
1956             :  * When pass the actual path, the caller should calculate credits
1957             :  * under i_data_sem.
1958             :  */
1959             : int ext4_ext_calc_credits_for_single_extent(struct inode *inode, int nrblocks,
1960             :                                             struct ext4_ext_path *path)
1961        0 : {
1962        0 :          if (path) {
1963        0 :                  int depth = ext_depth(inode);
1964        0 :                  int ret = 0;
1965             :
1966             :                  /* probably there is space in leaf? */
1967        0 :                  if (le16_to_cpu(path[depth].p_hdr->eh_entries)
1968             :                                  < le16_to_cpu(path[depth].p_hdr->eh_max)) {
1969             :
1970             :                          /*
1971             :                           *  There are some space in the leaf tree, no
1972             :                           *  need to account for leaf block credit
1973             :                           *
1974             :                           *  bitmaps and block group descriptor blocks
1975             :                           *  and other metadat blocks still need to be
```

```
1976                :                          *  accounted.
1977                :                          */
1978                :                         /* 1 bitmap, 1 block group descriptor */
1979              0 :                         ret = 2 + EXT4_META_TRANS_BLOCKS(inode->i_sb);
1980              0 :                         return ret;
1981                :                 }
1982                :         }
1983                :
1984              0 :         return ext4_chunk_trans_blocks(inode, nrblocks);
1985                : }
1986                :
1987                : /*
1988                :  * How many index/leaf blocks need to change/allocate to modify nrblocks?
1989                :  *
1990                :  * if nrblocks are fit in a single extent (chunk flag is 1), then
1991                :  * in the worse case, each tree level index/leaf need to be changed
1992                :  * if the tree split due to insert a new extent, then the old tree
1993                :  * index/leaf need to be updated too
1994                :  *
1995                :  * If the nrblocks are discontiguous, they could cause
1996                :  * the whole tree split more than once, but this is really rare.
1997                :  */
1998                : int ext4_ext_index_trans_blocks(struct inode *inode, int nrblocks, int chunk)
1999        245562518 : {
2000                :         int index;
2001        245562518 :         int depth = ext_depth(inode);
2002                :
2003        245562518 :         if (chunk)
2004         34743512 :                 index = depth * 2;
2005                :         else
2006        210819006 :                 index = depth * 3;
2007                :
2008        245562518 :         return index;
2009                : }
2010                :
2011                : static int ext4_remove_blocks(handle_t *handle, struct inode *inode,
2012                :                               struct ext4_extent *ex,
2013                :                               ext4_lblk_t from, ext4_lblk_t to)
2014          3680513 : {
2015                :         struct buffer_head *bh;
2016          3680513 :         unsigned short ee_len =  ext4_ext_get_actual_len(ex);
2017          3680513 :         int i, metadata = 0;
2018                :
2019          3680513 :         if (S_ISDIR(inode->i_mode) || S_ISLNK(inode->i_mode))
2020           655404 :                 metadata = 1;
2021                : #ifdef EXTENTS_STATS
2022                :         {
2023                :                 struct ext4_sb_info *sbi = EXT4_SB(inode->i_sb);
2024                :                 spin_lock(&sbi->s_ext_stats_lock);
2025                :                 sbi->s_ext_blocks += ee_len;
2026                :                 sbi->s_ext_extents++;
2027                :                 if (ee_len < sbi->s_ext_min)
2028                :                         sbi->s_ext_min = ee_len;
2029                :                 if (ee_len > sbi->s_ext_max)
2030                :                         sbi->s_ext_max = ee_len;
2031                :                 if (ext_depth(inode) > sbi->s_depth_max)
2032                :                         sbi->s_depth_max = ext_depth(inode);
2033                :                 spin_unlock(&sbi->s_ext_stats_lock);
2034                :         }
2035                : #endif
2036          7361029 :         if (from >= le32_to_cpu(ex->ee_block)
2037                :             && to == le32_to_cpu(ex->ee_block) + ee_len - 1) {
2038                :                 /* tail removal */
2039                :                 ext4_lblk_t num;
2040                :                 ext4_fsblk_t start;
2041                :
2042          3680769 :                 num = le32_to_cpu(ex->ee_block) + ee_len - from;
2043          3680769 :                 start = ext_pblock(ex) + ee_len - num;
2044                :                 ext_debug("free last %u blocks starting %llu\n", num, start);
2045        157219845 :                 for (i = 0; i < num; i++) {
2046        307078741 :                         bh = sb_find_get_block(inode->i_sb, start + i);
```

```
2047  153539412 :                         ext4_forget(handle, 0, inode, bh, start + i);
2048            :                 }
2049    3680516 :                 ext4_free_blocks(handle, inode, start, num, metadata);
2050          0 :         } else if (from == le32_to_cpu(ex->ee_block)
2051            :                    && to <= le32_to_cpu(ex->ee_block) + ee_len - 1) {
2052          0 :                 printk(KERN_INFO "strange request: removal %u-%u from %u:%u\n",
2053            :                         from, to, le32_to_cpu(ex->ee_block), ee_len);
2054            :         } else {
2055          0 :                 printk(KERN_INFO "strange request: removal(2) "
2056            :                                  "%u-%u from %u:%u\n",
2057            :                                  from, to, le32_to_cpu(ex->ee_block), ee_len);
2058            :         }
2059    3680516 :         return 0;
2060            : }
2061            :
2062            : static int
2063            : ext4_ext_rm_leaf(handle_t *handle, struct inode *inode,
2064            :                 struct ext4_ext_path *path, ext4_lblk_t start)
2065    1662896 : {
2066    1662896 :         int err = 0, correct_index = 0;
2067    1662896 :         int depth = ext_depth(inode), credits;
2068            :         struct ext4_extent_header *eh;
2069            :         ext4_lblk_t a, b, block;
2070            :         unsigned num;
2071            :         ext4_lblk_t ex_ee_block;
2072            :         unsigned short ex_ee_len;
2073    1662896 :         unsigned uninitialized = 0;
2074            :         struct ext4_extent *ex;
2075            :
2076            :         /* the header must be checked already in ext4_ext_remove_space() */
2077            :         ext_debug("truncate since %u in leaf\n", start);
2078    1662896 :         if (!path[depth].p_hdr)
2079     431524 :                 path[depth].p_hdr = ext_block_hdr(path[depth].p_bh);
2080    1662896 :         eh = path[depth].p_hdr;
2081    1662896 :         BUG_ON(eh == NULL);
2082            :
2083            :         /* find where to start removing */
2084    1662896 :         ex = EXT_LAST_EXTENT(eh);
2085            :
2086    1662896 :         ex_ee_block = le32_to_cpu(ex->ee_block);
2087    1662896 :         ex_ee_len = ext4_ext_get_actual_len(ex);
2088            :
2089    7006305 :         while (ex >= EXT_FIRST_EXTENT(eh) &&
2090            :                         ex_ee_block + ex_ee_len > start) {
2091            :
2092    3680514 :                 if (ext4_ext_is_uninitialized(ex))
2093          0 :                         uninitialized = 1;
2094            :                 else
2095    3680514 :                         uninitialized = 0;
2096            :
2097            :                 ext_debug("remove ext %lu:%u\n", ex_ee_block, ex_ee_len);
2098    3680514 :                 path[depth].p_ext = ex;
2099            :
2100    3680514 :                 a = ex_ee_block > start ? ex_ee_block : start;
2101    3680514 :                 b = ex_ee_block + ex_ee_len - 1 < EXT_MAX_BLOCK ?
2102            :                         ex_ee_block + ex_ee_len - 1 : EXT_MAX_BLOCK;
2103            :
2104            :                 ext_debug("  border %u:%u\n", a, b);
2105            :
2106    3680514 :                 if (a != ex_ee_block && b != ex_ee_block + ex_ee_len - 1) {
2107          0 :                         block = 0;
2108          0 :                         num = 0;
2109          0 :                         BUG();
2110    3680515 :                 } else if (a != ex_ee_block) {
2111            :                         /* remove tail of the extent */
2112          0 :                         block = ex_ee_block;
2113          0 :                         num = a - block;
2114    3680515 :                 } else if (b != ex_ee_block + ex_ee_len - 1) {
2115            :                         /* remove head of the extent */
2116          0 :                         block = a;
2117          0 :                         num = b - a;
```

```
2118                :                         /* there is no "make a hole" API yet */
2119              0 :                         BUG();
2120                :                 } else {
2121                :                         /* remove whole extent: excellent! */
2122        3680515 :                         block = ex_ee_block;
2123        3680515 :                         num = 0;
2124        3680515 :                         BUG_ON(a != ex_ee_block);
2125        3680515 :                         BUG_ON(b != ex_ee_block + ex_ee_len - 1);
2126                :                 }
2127                :
2128                :                 /*
2129                :                  * 3 for leaf, sb, and inode plus 2 (bmap and group
2130                :                  * descriptor) for each block group; assume two block
2131                :                  * groups plus ex_ee_len/blocks_per_block_group for
2132                :                  * the worst case
2133                :                  */
2134        7361032 :                 credits = 7 + 2*(ex_ee_len/EXT4_BLOCKS_PER_GROUP(inode->i_sb));
2135        3680516 :                 if (ex == EXT_FIRST_EXTENT(eh)) {
2136        1662886 :                         correct_index = 1;
2137        1662886 :                         credits += (ext_depth(inode)) + 1;
2138                :                 }
2139        7361032 :                 credits += 2 * EXT4_QUOTA_TRANS_BLOCKS(inode->i_sb);
2140                :
2141        3680516 :                 err = ext4_ext_journal_restart(handle, credits);
2142        3680516 :                 if (err)
2143              0 :                         goto out;
2144                :
2145        7361030 :                 err = ext4_ext_get_access(handle, inode, path + depth);
2146        3680514 :                 if (err)
2147              0 :                         goto out;
2148                :
2149        3680514 :                 err = ext4_remove_blocks(handle, inode, ex, a, b);
2150        3680515 :                 if (err)
2151              0 :                         goto out;
2152                :
2153        3680515 :                 if (num == 0) {
2154                :                         /* this extent is removed; mark slot entirely unused */
2155                :                         ext4_ext_store_pblock(ex, 0);
2156        3680514 :                         le16_add_cpu(&eh->eh_entries, -1);
2157                :                 }
2158                :
2159        3680515 :                 ex->ee_block = cpu_to_le32(block);
2160        3680515 :                 ex->ee_len = cpu_to_le16(num);
2161                :                 /*
2162                :                  * Do not mark uninitialized if all the blocks in the
2163                :                  * extent have been removed.
2164                :                  */
2165        3680515 :                 if (uninitialized && num)
2166                :                         ext4_ext_mark_uninitialized(ex);
2167                :
2168        3680515 :                 err = ext4_ext_dirty(handle, inode, path + depth);
2169        3680513 :                 if (err)
2170              0 :                         goto out;
2171                :
2172                :                 ext_debug("new extent: %u:%u:%llu\n", block, num,
2173                :                                 ext_pblock(ex));
2174        3680513 :                 ex--;
2175        3680513 :                 ex_ee_block = le32_to_cpu(ex->ee_block);
2176        3680513 :                 ex_ee_len = ext4_ext_get_actual_len(ex);
2177                :         }
2178                :
2179        1662895 :         if (correct_index && eh->eh_entries)
2180              0 :                 err = ext4_ext_correct_indexes(handle, inode, path);
2181                :
2182                :         /* if this leaf is free, then we should
2183                :          * remove it from index block above */
2184        1662895 :         if (err == 0 && eh->eh_entries == 0 && path[depth].p_bh != NULL)
2185         215753 :                 err = ext4_ext_rm_idx(handle, inode, path + depth);
2186                :
2187        1662895 : out:
2188        1662895 :         return err;
```

```
2189                :  }
2190                :
2191                :  /*
2192                :   * ext4_ext_more_to_rm:
2193                :   * returns 1 if current index has to be freed (even partial)
2194                :   */
2195                :  static int
2196                :  ext4_ext_more_to_rm(struct ext4_ext_path *path)
2197                :  {
2198          431528 :          BUG_ON(path->p_idx == NULL);
2199                :
2200          431527 :          if (path->p_idx < EXT_FIRST_INDEX(path->p_hdr))
2201          215761 :                  return 0;
2202                :
2203                :          /*
2204                :           * if truncate on deeper level happened, it wasn't partial,
2205                :           * so we have to consider current index for truncation
2206                :           */
2207          215766 :          if (le16_to_cpu(path->p_hdr->eh_entries) == path->p_block)
2208               2 :                  return 0;
2209          215764 :          return 1;
2210                :  }
2211                :
2212                :  static int ext4_ext_remove_space(struct inode *inode, ext4_lblk_t start)
2213         1662897 :  {
2214         1662897 :          struct super_block *sb = inode->i_sb;
2215         1662897 :          int depth = ext_depth(inode);
2216                :          struct ext4_ext_path *path;
2217                :          handle_t *handle;
2218         1662897 :          int i = 0, err = 0;
2219                :
2220                :          ext_debug("truncate since %u\n", start);
2221                :
2222                :          /* probably first extent we're gonna free will be last in block */
2223         3325794 :          handle = ext4_journal_start(inode, depth + 1);
2224         1662897 :          if (IS_ERR(handle))
2225               0 :                  return PTR_ERR(handle);
2226                :
2227                :          ext4_ext_invalidate_cache(inode);
2228                :
2229                :          /*
2230                :           * We start scanning from right side, freeing all the blocks
2231                :           * after i_size and walking into the tree depth-wise.
2232                :           */
2233         1662897 :          path = kzalloc(sizeof(struct ext4_ext_path) * (depth + 1), GFP_NOFS);
2234         1662897 :          if (path == NULL) {
2235               0 :                  ext4_journal_stop(handle);
2236               0 :                  return -ENOMEM;
2237                :          }
2238         1662897 :          path[0].p_hdr = ext_inode_hdr(inode);
2239         1662897 :          if (ext4_ext_check(inode, path[0].p_hdr, depth)) {
2240               0 :                  err = -EIO;
2241               0 :                  goto out;
2242                :          }
2243         1662897 :          path[0].p_depth = depth;
2244                :
2245         5420211 :          while (i >= 0 && err == 0) {
2246         2094425 :                  if (i == depth) {
2247                :                          /* this is leaf block */
2248         1662897 :                          err = ext4_ext_rm_leaf(handle, inode, path, start);
2249                :                          /* root level has p_bh == NULL, brelse() eats this */
2250         1662897 :                          brelse(path[i].p_bh);
2251         1662889 :                          path[i].p_bh = NULL;
2252         1662889 :                          i--;
2253         1662889 :                          continue;
2254                :                  }
2255                :
2256                :                  /* this is index block */
2257          431528 :                  if (!path[i].p_hdr) {
2258                :                          ext_debug("initialize header\n");
2259               4 :                          path[i].p_hdr = ext_block_hdr(path[i].p_bh);
```

```
2260                :                 }
2261                :
2262         431528 :                 if (!path[i].p_idx) {
2263                :                         /* this level hasn't been touched yet */
2264         215764 :                         path[i].p_idx = EXT_LAST_INDEX(path[i].p_hdr);
2265         215764 :                         path[i].p_block = le16_to_cpu(path[i].p_hdr->eh_entries)+1;
2266                :                         ext_debug("init index ptr: hdr 0x%p, num %d\n",
2267                :                                   path[i].p_hdr,
2268                :                                   le16_to_cpu(path[i].p_hdr->eh_entries));
2269                :                 } else {
2270                :                         /* we were already here, see at next index */
2271         215764 :                         path[i].p_idx--;
2272                :                 }
2273                :
2274                :                 ext_debug("level %d - index, first 0x%p, cur 0x%p\n",
2275                :                                 i, EXT_FIRST_INDEX(path[i].p_hdr),
2276                :                                 path[i].p_idx);
2277         863055 :                 if (ext4_ext_more_to_rm(path + i)) {
2278                :                         struct buffer_head *bh;
2279                :                         /* go to the next level */
2280                :                         ext_debug("move to level %d (block %llu)\n",
2281                :                                   i + 1, idx_pblock(path[i].p_idx));
2282         215763 :                         memset(path + i + 1, 0, sizeof(*path));
2283         431526 :                         bh = sb_bread(sb, idx_pblock(path[i].p_idx));
2284         215763 :                         if (!bh) {
2285                :                                 /* should we reset i_size? */
2286              0 :                                 err = -EIO;
2287              0 :                                 break;
2288                :                         }
2289         215763 :                         if (WARN_ON(i + 1 > depth)) {
2290              0 :                                 err = -EIO;
2291              0 :                                 break;
2292                :                         }
2293         431528 :                         if (ext4_ext_check(inode, ext_block_hdr(bh),
2294                :                                                         depth - i - 1)) {
2295              0 :                                 err = -EIO;
2296              0 :                                 break;
2297                :                         }
2298         215764 :                         path[i + 1].p_bh = bh;
2299                :
2300                :                         /* save actual number of indexes since this
2301                :                          * number is changed at the next iteration */
2302         215764 :                         path[i].p_block = le16_to_cpu(path[i].p_hdr->eh_entries);
2303         215764 :                         i++;
2304                :                 } else {
2305                :                         /* we finished processing this index, go up */
2306         215764 :                         if (path[i].p_hdr->eh_entries == 0 && i > 0) {
2307                :                                 /* index is empty, remove it;
2308                :                                  * handle must be already prepared by the
2309                :                                  * truncatei_leaf() */
2310              0 :                                 err = ext4_ext_rm_idx(handle, inode, path + i);
2311                :                         }
2312                :                         /* root level has p_bh == NULL, brelse() eats this */
2313         215764 :                         brelse(path[i].p_bh);
2314         215764 :                         path[i].p_bh = NULL;
2315         215764 :                         i--;
2316                :                         ext_debug("return to level %d\n", i);
2317                :                 }
2318                :         }
2319                :
2320                :         /* TODO: flexible tree reduction should be here */
2321        1662889 :         if (path->p_hdr->eh_entries == 0) {
2322                :                 /*
2323                :                  * truncate to zero freed all the tree,
2324                :                  * so we need to correct eh_depth
2325                :                  */
2326        1662884 :                 err = ext4_ext_get_access(handle, inode, path);
2327        1662884 :                 if (err == 0) {
2328        1662885 :                         ext_inode_hdr(inode)->eh_depth = 0;
2329        1662885 :                         ext_inode_hdr(inode)->eh_max =
2330                :                                 cpu_to_le16(ext4_ext_space_root(inode));
```

```
2331  1662885 :                         err = ext4_ext_dirty(handle, inode, path);
2332          :                 }
2333          :         }
2334  1662890 : out:
2335  1662890 :         ext4_ext_drop_refs(path);
2336  1662897 :         kfree(path);
2337  1662892 :         ext4_journal_stop(handle);
2338          :
2339  1662895 :         return err;
2340          : }
2341          :
2342          : /*
2343          :  * called at mount time
2344          :  */
2345          : void ext4_ext_init(struct super_block *sb)
2346       94 : {
2347          :         /*
2348          :          * possible initialization would be here
2349          :          */
2350          :
2351       94 :         if (EXT4_HAS_INCOMPAT_FEATURE(sb, EXT4_FEATURE_INCOMPAT_EXTENTS)) {
2352       77 :                 printk(KERN_INFO "EXT4-fs: file extents enabled");
2353          : #ifdef AGGRESSIVE_TEST
2354          :                 printk(", aggressive tests");
2355          : #endif
2356          : #ifdef CHECK_BINSEARCH
2357          :                 printk(", check binsearch");
2358          : #endif
2359          : #ifdef EXTENTS_STATS
2360          :                 printk(", stats");
2361          : #endif
2362       77 :                 printk("\n");
2363          : #ifdef EXTENTS_STATS
2364          :                 spin_lock_init(&EXT4_SB(sb)->s_ext_stats_lock);
2365          :                 EXT4_SB(sb)->s_ext_min = 1 << 30;
2366          :                 EXT4_SB(sb)->s_ext_max = 0;
2367          : #endif
2368          :         }
2369       94 : }
2370          :
2371          : /*
2372          :  * called at umount time
2373          :  */
2374          : void ext4_ext_release(struct super_block *sb)
2375       94 : {
2376       94 :         if (!EXT4_HAS_INCOMPAT_FEATURE(sb, EXT4_FEATURE_INCOMPAT_EXTENTS))
2377          :                 return;
2378          :
2379          : #ifdef EXTENTS_STATS
2380          :         if (EXT4_SB(sb)->s_ext_blocks && EXT4_SB(sb)->s_ext_extents) {
2381          :                 struct ext4_sb_info *sbi = EXT4_SB(sb);
2382          :                 printk(KERN_ERR "EXT4-fs: %lu blocks in %lu extents (%lu ave)\n",
2383          :                         sbi->s_ext_blocks, sbi->s_ext_extents,
2384          :                         sbi->s_ext_blocks / sbi->s_ext_extents);
2385          :                 printk(KERN_ERR "EXT4-fs: extents: %lu min, %lu max, max depth %lu\n",
2386          :                         sbi->s_ext_min, sbi->s_ext_max, sbi->s_depth_max);
2387          :         }
2388          : #endif
2389          : }
2390          :
2391          : static void bi_complete(struct bio *bio, int error)
2392       10 : {
2393       10 :         complete((struct completion *)bio->bi_private);
2394       10 : }
2395          :
2396          : /* FIXME!! we need to try to merge to left or right after zero-out  */
2397          : static int ext4_ext_zeroout(struct inode *inode, struct ext4_extent *ex)
2398       10 : {
2399       10 :         int ret = -EIO;
2400          :         struct bio *bio;
2401          :         int blkbits, blocksize;
```

```
2402                :         sector_t ee_pblock;
2403                :         struct completion event;
2404                :         unsigned int ee_len, len, done, offset;
2405                :
2406                :
2407               10 :         blkbits   = inode->i_blkbits;
2408               10 :         blocksize = inode->i_sb->s_blocksize;
2409               10 :         ee_len    = ext4_ext_get_actual_len(ex);
2410               10 :         ee_pblock = ext_pblock(ex);
2411                :
2412                :         /* convert ee_pblock to 512 byte sectors */
2413               10 :         ee_pblock = ee_pblock << (blkbits - 9);
2414                :
2415               30 :         while (ee_len > 0) {
2416                :
2417               10 :                 if (ee_len > BIO_MAX_PAGES)
2418                0 :                         len = BIO_MAX_PAGES;
2419                :                 else
2420               10 :                         len = ee_len;
2421                :
2422               10 :                 bio = bio_alloc(GFP_NOIO, len);
2423               10 :                 bio->bi_sector = ee_pblock;
2424               10 :                 bio->bi_bdev   = inode->i_sb->s_bdev;
2425                :
2426               10 :                 done = 0;
2427               10 :                 offset = 0;
2428               32 :                 while (done < len) {
2429               12 :                         ret = bio_add_page(bio, ZERO_PAGE(0),
2430                :                                                         blocksize, offset);
2431               12 :                         if (ret != blocksize) {
2432                :                                 /*
2433                :                                  * We can't add any more pages because of
2434                :                                  * hardware limitations.  Start a new bio.
2435                :                                  */
2436                0 :                                 break;
2437                :                         }
2438               12 :                         done++;
2439               12 :                         offset += blocksize;
2440               12 :                         if (offset >= PAGE_CACHE_SIZE)
2441                6 :                                 offset = 0;
2442                :                 }
2443                :
2444                :                 init_completion(&event);
2445               10 :                 bio->bi_private = &event;
2446               10 :                 bio->bi_end_io = bi_complete;
2447               10 :                 submit_bio(WRITE, bio);
2448               10 :                 wait_for_completion(&event);
2449                :
2450               20 :                 if (test_bit(BIO_UPTODATE, &bio->bi_flags))
2451               10 :                         ret = 0;
2452                :                 else {
2453                0 :                         ret = -EIO;
2454                0 :                         break;
2455                :                 }
2456               10 :                 bio_put(bio);
2457               10 :                 ee_len    -= done;
2458               10 :                 ee_pblock += done  << (blkbits - 9);
2459                :         }
2460               10 :         return ret;
2461                : }
2462                :
2463                : #define EXT4_EXT_ZERO_LEN 7
2464                :
2465                : /*
2466                :  * This function is called by ext4_ext_get_blocks() if someone tries to write
2467                :  * to an uninitialized extent. It may result in splitting the uninitialized
2468                :  * extent into multiple extents (upto three - one initialized and two
2469                :  * uninitialized).
2470                :  * There are three possibilities:
2471                :  *   a> There is no split required: Entire extent should be initialized
2472                :  *   b> Splits in two extents: Write is happening at either end of the extent
```

```
2473                :  *   c> Splits in three extents: Somone is writing in middle of the extent
2474                :  */
2475                : static int ext4_ext_convert_to_initialized(handle_t *handle,
2476                :                                           struct inode *inode,
2477                :                                           struct ext4_ext_path *path,
2478                :                                           ext4_lblk_t iblock,
2479                :                                           unsigned int max_blocks)
2480             10 : {
2481                :         struct ext4_extent *ex, newex, orig_ex;
2482             10 :         struct ext4_extent *ex1 = NULL;
2483             10 :         struct ext4_extent *ex2 = NULL;
2484             10 :         struct ext4_extent *ex3 = NULL;
2485                :         struct ext4_extent_header *eh;
2486                :         ext4_lblk_t ee_block;
2487                :         unsigned int allocated, ee_len, depth;
2488                :         ext4_fsblk_t newblock;
2489             10 :         int err = 0;
2490             10 :         int ret = 0;
2491                :
2492             10 :         depth = ext_depth(inode);
2493             10 :         eh = path[depth].p_hdr;
2494             10 :         ex = path[depth].p_ext;
2495             10 :         ee_block = le32_to_cpu(ex->ee_block);
2496             10 :         ee_len = ext4_ext_get_actual_len(ex);
2497             10 :         allocated = ee_len - (iblock - ee_block);
2498             20 :         newblock = iblock - ee_block + ext_pblock(ex);
2499             10 :         ex2 = ex;
2500             10 :         orig_ex.ee_block = ex->ee_block;
2501             10 :         orig_ex.ee_len   = cpu_to_le16(ee_len);
2502                :         ext4_ext_store_pblock(&orig_ex, ext_pblock(ex));
2503                :
2504             20 :         err = ext4_ext_get_access(handle, inode, path + depth);
2505             10 :         if (err)
2506              0 :                 goto out;
2507                :         /* If extent has less than 2*EXT4_EXT_ZERO_LEN zerout directly */
2508             10 :         if (ee_len <= 2*EXT4_EXT_ZERO_LEN) {
2509             10 :                 err =  ext4_ext_zeroout(inode, &orig_ex);
2510             10 :                 if (err)
2511              0 :                         goto fix_extent_len;
2512                :                 /* update the extent length and mark as initialized */
2513             10 :                 ex->ee_block = orig_ex.ee_block;
2514             10 :                 ex->ee_len   = orig_ex.ee_len;
2515                :                 ext4_ext_store_pblock(ex, ext_pblock(&orig_ex));
2516             10 :                 ext4_ext_dirty(handle, inode, path + depth);
2517                :                 /* zeroed the full extent */
2518             10 :                 return allocated;
2519                :         }
2520                :
2521                :         /* ex1: ee_block to iblock - 1 : uninitialized */
2522              0 :         if (iblock > ee_block) {
2523              0 :                 ex1 = ex;
2524              0 :                 ex1->ee_len = cpu_to_le16(iblock - ee_block);
2525                :                 ext4_ext_mark_uninitialized(ex1);
2526              0 :                 ex2 = &newex;
2527                :         }
2528                :         /*
2529                :          * for sanity, update the length of the ex2 extent before
2530                :          * we insert ex3, if ex1 is NULL. This is to avoid temporary
2531                :          * overlap of blocks.
2532                :          */
2533              0 :         if (!ex1 && allocated > max_blocks)
2534              0 :                 ex2->ee_len = cpu_to_le16(max_blocks);
2535                :         /* ex3: to ee_block + ee_len : uninitialised */
2536              0 :         if (allocated > max_blocks) {
2537                :                 unsigned int newdepth;
2538                :                 /* If extent has less than EXT4_EXT_ZERO_LEN zerout directly */
2539              0 :                 if (allocated <= EXT4_EXT_ZERO_LEN) {
2540                :                         /*
2541                :                          * iblock == ee_block is handled by the zerouout
2542                :                          * at the beginning.
2543                :                          * Mark first half uninitialized.
```

```
2544                :                          * Mark second half initialized and zero out the
2545                :                          * initialized extent
2546                :                          */
2547              0 :                         ex->ee_block = orig_ex.ee_block;
2548              0 :                         ex->ee_len   = cpu_to_le16(ee_len - allocated);
2549                :                         ext4_ext_mark_uninitialized(ex);
2550                :                         ext4_ext_store_pblock(ex, ext_pblock(&orig_ex));
2551              0 :                         ext4_ext_dirty(handle, inode, path + depth);
2552                :
2553              0 :                         ex3 = &newex;
2554              0 :                         ex3->ee_block = cpu_to_le32(iblock);
2555                :                         ext4_ext_store_pblock(ex3, newblock);
2556              0 :                         ex3->ee_len = cpu_to_le16(allocated);
2557              0 :                         err = ext4_ext_insert_extent(handle, inode, path, ex3);
2558              0 :                         if (err == -ENOSPC) {
2559              0 :                                 err =  ext4_ext_zeroout(inode, &orig_ex);
2560              0 :                                 if (err)
2561              0 :                                         goto fix_extent_len;
2562              0 :                                 ex->ee_block = orig_ex.ee_block;
2563              0 :                                 ex->ee_len   = orig_ex.ee_len;
2564                :                                 ext4_ext_store_pblock(ex, ext_pblock(&orig_ex));
2565              0 :                                 ext4_ext_dirty(handle, inode, path + depth);
2566                :                                 /* blocks available from iblock */
2567              0 :                                 return allocated;
2568                :
2569              0 :                         } else if (err)
2570              0 :                                 goto fix_extent_len;
2571                :
2572                :                         /*
2573                :                          * We need to zero out the second half because
2574                :                          * an fallocate request can update file size and
2575                :                          * converting the second half to initialized extent
2576                :                          * implies that we can leak some junk data to user
2577                :                          * space.
2578                :                          */
2579              0 :                         err =  ext4_ext_zeroout(inode, ex3);
2580              0 :                         if (err) {
2581                :                                 /*
2582                :                                  * We should actually mark the
2583                :                                  * second half as uninit and return error
2584                :                                  * Insert would have changed the extent
2585                :                                  */
2586              0 :                                 depth = ext_depth(inode);
2587              0 :                                 ext4_ext_drop_refs(path);
2588              0 :                                 path = ext4_ext_find_extent(inode,
2589                :                                                             iblock, path);
2590              0 :                                 if (IS_ERR(path)) {
2591              0 :                                         err = PTR_ERR(path);
2592              0 :                                         return err;
2593                :                                 }
2594                :                                 /* get the second half extent details */
2595              0 :                                 ex = path[depth].p_ext;
2596              0 :                                 err = ext4_ext_get_access(handle, inode,
2597                :                                                           path + depth);
2598              0 :                                 if (err)
2599              0 :                                         return err;
2600                :                                 ext4_ext_mark_uninitialized(ex);
2601              0 :                                 ext4_ext_dirty(handle, inode, path + depth);
2602              0 :                                 return err;
2603                :                         }
2604                :
2605                :                         /* zeroed the second half */
2606              0 :                         return allocated;
2607                :                 }
2608              0 :                 ex3 = &newex;
2609              0 :                 ex3->ee_block = cpu_to_le32(iblock + max_blocks);
2610              0 :                 ext4_ext_store_pblock(ex3, newblock + max_blocks);
2611              0 :                 ex3->ee_len = cpu_to_le16(allocated - max_blocks);
2612                :                 ext4_ext_mark_uninitialized(ex3);
2613              0 :                 err = ext4_ext_insert_extent(handle, inode, path, ex3);
2614              0 :                 if (err == -ENOSPC) {
```

```
2615       0 :                         err =  ext4_ext_zeroout(inode, &orig_ex);
2616       0 :                         if (err)
2617       0 :                                 goto fix_extent_len;
2618         :                         /* update the extent length and mark as initialized */
2619       0 :                         ex->ee_block = orig_ex.ee_block;
2620       0 :                         ex->ee_len   = orig_ex.ee_len;
2621         :                         ext4_ext_store_pblock(ex, ext_pblock(&orig_ex));
2622       0 :                         ext4_ext_dirty(handle, inode, path + depth);
2623         :                         /* zeroed the full extent */
2624         :                         /* blocks available from iblock */
2625       0 :                         return allocated;
2626         :
2627       0 :                 } else if (err)
2628       0 :                         goto fix_extent_len;
2629         :                 /*
2630         :                  * The depth, and hence eh & ex might change
2631         :                  * as part of the insert above.
2632         :                  */
2633       0 :                 newdepth = ext_depth(inode);
2634         :                 /*
2635         :                  * update the extent length after successful insert of the
2636         :                  * split extent
2637         :                  */
2638       0 :                 orig_ex.ee_len = cpu_to_le16(ee_len -
2639         :                                                 ext4_ext_get_actual_len(ex3));
2640       0 :                 depth = newdepth;
2641       0 :                 ext4_ext_drop_refs(path);
2642       0 :                 path = ext4_ext_find_extent(inode, iblock, path);
2643       0 :                 if (IS_ERR(path)) {
2644       0 :                         err = PTR_ERR(path);
2645       0 :                         goto out;
2646         :                 }
2647       0 :                 eh = path[depth].p_hdr;
2648       0 :                 ex = path[depth].p_ext;
2649       0 :                 if (ex2 != &newex)
2650       0 :                         ex2 = ex;
2651         :
2652       0 :                 err = ext4_ext_get_access(handle, inode, path + depth);
2653       0 :                 if (err)
2654       0 :                         goto out;
2655         :
2656       0 :                 allocated = max_blocks;
2657         :
2658         :                 /* If extent has less than EXT4_EXT_ZERO_LEN and we are trying
2659         :                  * to insert a extent in the middle zerout directly
2660         :                  * otherwise give the extent a chance to merge to left
2661         :                  */
2662       0 :                 if (le16_to_cpu(orig_ex.ee_len) <= EXT4_EXT_ZERO_LEN &&
2663         :                                                         iblock != ee_block) {
2664       0 :                         err =  ext4_ext_zeroout(inode, &orig_ex);
2665       0 :                         if (err)
2666       0 :                                 goto fix_extent_len;
2667         :                         /* update the extent length and mark as initialized */
2668       0 :                         ex->ee_block = orig_ex.ee_block;
2669       0 :                         ex->ee_len   = orig_ex.ee_len;
2670         :                         ext4_ext_store_pblock(ex, ext_pblock(&orig_ex));
2671       0 :                         ext4_ext_dirty(handle, inode, path + depth);
2672         :                         /* zero out the first half */
2673         :                         /* blocks available from iblock */
2674       0 :                         return allocated;
2675         :                 }
2676         :         }
2677         :         /*
2678         :          * If there was a change of depth as part of the
2679         :          * insertion of ex3 above, we need to update the length
2680         :          * of the ex1 extent again here
2681         :          */
2682       0 :         if (ex1 && ex1 != ex) {
2683       0 :                 ex1 = ex;
2684       0 :                 ex1->ee_len = cpu_to_le16(iblock - ee_block);
2685         :                 ext4_ext_mark_uninitialized(ex1);
```

```
2686                0 :                         ex2 = &newex;
2687                  :                 }
2688                  :                 /* ex2: iblock to iblock + maxblocks-1 : initialised */
2689                0 :                 ex2->ee_block = cpu_to_le32(iblock);
2690                  :                 ext4_ext_store_pblock(ex2, newblock);
2691                0 :                 ex2->ee_len = cpu_to_le16(allocated);
2692                0 :                 if (ex2 != ex)
2693                0 :                         goto insert;
2694                  :                 /*
2695                  :                  * New (initialized) extent starts from the first block
2696                  :                  * in the current extent. i.e., ex2 == ex
2697                  :                  * We have to see if it can be merged with the extent
2698                  :                  * on the left.
2699                  :                  */
2700                0 :                 if (ex2 > EXT_FIRST_EXTENT(eh)) {
2701                  :                         /*
2702                  :                          * To merge left, pass "ex2 - 1" to try_to_merge(),
2703                  :                          * since it merges towards right _only_.
2704                  :                          */
2705                0 :                         ret = ext4_ext_try_to_merge(inode, path, ex2 - 1);
2706                0 :                         if (ret) {
2707                0 :                                 err = ext4_ext_correct_indexes(handle, inode, path);
2708                0 :                                 if (err)
2709                0 :                                         goto out;
2710                0 :                                 depth = ext_depth(inode);
2711                0 :                                 ex2--;
2712                  :                         }
2713                  :                 }
2714                  :                 /*
2715                  :                  * Try to Merge towards right. This might be required
2716                  :                  * only when the whole extent is being written to.
2717                  :                  * i.e. ex2 == ex and ex3 == NULL.
2718                  :                  */
2719                0 :                 if (!ex3) {
2720                0 :                         ret = ext4_ext_try_to_merge(inode, path, ex2);
2721                0 :                         if (ret) {
2722                0 :                                 err = ext4_ext_correct_indexes(handle, inode, path);
2723                0 :                                 if (err)
2724                0 :                                         goto out;
2725                  :                         }
2726                  :                 }
2727                  :                 /* Mark modified extent as dirty */
2728                0 :                 err = ext4_ext_dirty(handle, inode, path + depth);
2729                0 :                 goto out;
2730                0 : insert:
2731                0 :                 err = ext4_ext_insert_extent(handle, inode, path, &newex);
2732                0 :                 if (err == -ENOSPC) {
2733                0 :                         err =  ext4_ext_zeroout(inode, &orig_ex);
2734                0 :                         if (err)
2735                0 :                                 goto fix_extent_len;
2736                  :                         /* update the extent length and mark as initialized */
2737                0 :                         ex->ee_block = orig_ex.ee_block;
2738                0 :                         ex->ee_len   = orig_ex.ee_len;
2739                  :                         ext4_ext_store_pblock(ex, ext_pblock(&orig_ex));
2740                0 :                         ext4_ext_dirty(handle, inode, path + depth);
2741                  :                         /* zero out the first half */
2742                0 :                         return allocated;
2743                0 :                 } else if (err)
2744                0 :                         goto fix_extent_len;
2745                0 : out:
2746                0 :                 return err ? err : allocated;
2747                  :
2748                0 : fix_extent_len:
2749                0 :                 ex->ee_block = orig_ex.ee_block;
2750                0 :                 ex->ee_len   = orig_ex.ee_len;
2751                  :                 ext4_ext_store_pblock(ex, ext_pblock(&orig_ex));
2752                  :                 ext4_ext_mark_uninitialized(ex);
2753                0 :                 ext4_ext_dirty(handle, inode, path + depth);
2754                0 :                 return err;
2755                  : }
2756                  :
```

```
2757                :  /*
2758                :   * Block allocation/map/preallocation routine for extents based files
2759                :   *
2760                :   *
2761                :   * Need to be called with
2762                :   * down_read(&EXT4_I(inode)->i_data_sem) if not allocating file system block
2763                :   * (ie, create is zero). Otherwise down_write(&EXT4_I(inode)->i_data_sem)
2764                :   *
2765                :   * return > 0, number of of blocks already mapped/allocated
2766                :   *          if create == 0 and these are pre-allocated blocks
2767                :   *              buffer head is unmapped
2768                :   *          otherwise blocks are mapped
2769                :   *
2770                :   * return = 0, if plain look up failed (blocks have not been allocated)
2771                :   *          buffer head is unmapped
2772                :   *
2773                :   * return < 0, error case.
2774                :   */
2775                :  int ext4_ext_get_blocks(handle_t *handle, struct inode *inode,
2776                :                          ext4_lblk_t iblock,
2777                :                          unsigned int max_blocks, struct buffer_head *bh_result,
2778                :                          int flags)
2779      1325108332 :  {
2780      1325108332 :          struct ext4_ext_path *path = NULL;
2781                 :          struct ext4_extent_header *eh;
2782                 :          struct ext4_extent newex, *ex;
2783                 :          ext4_fsblk_t newblock;
2784      1325108332 :          int err = 0, depth, ret, cache_type;
2785      1325108332 :          unsigned int allocated = 0;
2786                 :          struct ext4_allocation_request ar;
2787                 :
2788      1325108332 :          __clear_bit(BH_New, &bh_result->b_state);
2789                 :          ext_debug("blocks %u/%u requested for inode %u\n",
2790                 :                          iblock, max_blocks, inode->i_ino);
2791                 :
2792                 :          /* check in cache */
2793      1348151084 :          cache_type = ext4_ext_in_cache(inode, iblock, &newex);
2794      1348151084 :          if (cache_type) {
2795      1077355580 :                  if (cache_type == EXT4_EXT_CACHE_GAP) {
2796      1030972528 :                          if ((flags & EXT4_GET_BLOCKS_CREATE) == 0) {
2797                 :                                  /*
2798                 :                                   * block isn't allocated yet and
2799                 :                                   * user doesn't want to allocate it
2800                 :                                   */
2801       809248841 :                                  goto out2;
2802                 :                          }
2803                 :                          /* we should allocate requested block */
2804        46383052 :                  } else if (cache_type == EXT4_EXT_CACHE_EXTENT) {
2805                 :                          /* block is already allocated */
2806        92766104 :                          newblock = iblock
2807                 :                                  - le32_to_cpu(newex.ee_block)
2808                 :                                  + ext_pblock(&newex);
2809                 :                          /* number of remaining blocks in the extent */
2810        46383052 :                          allocated = ext4_ext_get_actual_len(&newex) -
2811                 :                                  (iblock - le32_to_cpu(newex.ee_block));
2812        46383052 :                          goto out;
2813                 :                  } else {
2814               0 :                          BUG();
2815                 :                  }
2816                 :          }
2817                 :
2818                 :          /* find extent for this block */
2819       492519191 :          path = ext4_ext_find_extent(inode, iblock, NULL);
2820       493584686 :          if (IS_ERR(path)) {
2821               0 :                  err = PTR_ERR(path);
2822               0 :                  path = NULL;
2823               0 :                  goto out2;
2824                 :          }
2825                 :
2826       493584686 :          depth = ext_depth(inode);
2827                 :
```

```
2828                :         /*
2829                :          * consistent leaf must not be empty;
2830                :          * this situation is possible, though, _during_ tree modification;
2831                :          * this is why assert can't be put in ext4_ext_find_extent()
2832                :          */
2833     493584686  :         BUG_ON(path[depth].p_ext == NULL && depth != 0);
2834     493071536  :         eh = path[depth].p_hdr;
2835                :
2836     493071536  :         ex = path[depth].p_ext;
2837     493071536  :         if (ex) {
2838     474733190  :                 ext4_lblk_t ee_block = le32_to_cpu(ex->ee_block);
2839     474733190  :                 ext4_fsblk_t ee_start = ext_pblock(ex);
2840                :                 unsigned short ee_len;
2841                :
2842                :                 /*
2843                :                  * Uninitialized extents are treated as holes, except that
2844                :                  * we split out initialized portions during a write.
2845                :                  */
2846     474733190  :                 ee_len = ext4_ext_get_actual_len(ex);
2847                :                 /* if found extent covers block, simply return it */
2848     474733190  :                 if (iblock >= ee_block && iblock < ee_block + ee_len) {
2849      52089640  :                         newblock = iblock - ee_block + ee_start;
2850                :                         /* number of remaining blocks in the extent */
2851      52089640  :                         allocated = ee_len - (iblock - ee_block);
2852                :                         ext_debug("%u fit into %lu:%d -> %llu\n", iblock,
2853                :                                         ee_block, ee_len, newblock);
2854                :
2855                :                         /* Do not put uninitialized extent in the cache */
2856      52089640  :                         if (!ext4_ext_is_uninitialized(ex)) {
2857      52089608  :                                 ext4_ext_put_in_cache(inode, ee_block,
2858                :                                                       ee_len, ee_start,
2859                :                                                       EXT4_EXT_CACHE_EXTENT);
2860      52089453  :                                 goto out;
2861                :                         }
2862            32  :                         if (flags & EXT4_GET_BLOCKS_UNINIT_EXT)
2863             4  :                                 goto out;
2864            28  :                         if ((flags & EXT4_GET_BLOCKS_CREATE) == 0) {
2865            18  :                                 if (allocated > max_blocks)
2866             0  :                                         allocated = max_blocks;
2867                :                                 /*
2868                :                                  * We have blocks reserved already.  We
2869                :                                  * return allocated blocks so that delalloc
2870                :                                  * won't do block reservation for us.  But
2871                :                                  * the buffer head will be unmapped so that
2872                :                                  * a read from the block returns 0s.
2873                :                                  */
2874                :                                 set_buffer_unwritten(bh_result);
2875            18  :                                 bh_result->b_bdev = inode->i_sb->s_bdev;
2876            18  :                                 bh_result->b_blocknr = newblock;
2877            18  :                                 goto out2;
2878                :                         }
2879                :
2880            10  :                         ret = ext4_ext_convert_to_initialized(handle, inode,
2881                :                                                               path, iblock,
2882                :                                                               max_blocks);
2883            10  :                         if (ret <= 0) {
2884             0  :                                 err = ret;
2885             0  :                                 goto out2;
2886                :                         } else
2887            10  :                                 allocated = ret;
2888            10  :                         goto outnew;
2889                :                 }
2890                :         }
2891                :
2892                :         /*
2893                :          * requested block isn't allocated yet;
2894                :          * we couldn't try to create block if create flag is zero
2895                :          */
2896     440981896  :         if ((flags & EXT4_GET_BLOCKS_CREATE) == 0) {
2897                :                 /*
2898                :                  * put just found gap into cache to speed up
```

```
2899                :                  * subsequent requests
2900                :                  */
2901                :                 ext4_ext_put_gap_in_cache(inode, path, iblock);
2902      221041207 :                 goto out2;
2903                :         }
2904                :         /*
2905                :          * Okay, we need to do block allocation.
2906                :          */
2907                :
2908                :         /* find neighbour allocated blocks */
2909      221114895 :         ar.lleft = iblock;
2910      221114895 :         err = ext4_ext_search_left(inode, path, &ar.lleft, &ar.pleft);
2911      220101218 :         if (err)
2912              0 :                 goto out2;
2913      220101218 :         ar.lright = iblock;
2914      220101218 :         err = ext4_ext_search_right(inode, path, &ar.lright, &ar.pright);
2915      220995594 :         if (err)
2916              0 :                 goto out2;
2917                :
2918                :         /*
2919                :          * See if request is beyond maximum number of blocks we can have in
2920                :          * a single extent. For an initialized extent this limit is
2921                :          * EXT_INIT_MAX_LEN and for an uninitialized extent this limit is
2922                :          * EXT_UNINIT_MAX_LEN.
2923                :          */
2924      220995594 :         if (max_blocks > EXT_INIT_MAX_LEN &&
2925                :             !(flags & EXT4_GET_BLOCKS_UNINIT_EXT))
2926              0 :                 max_blocks = EXT_INIT_MAX_LEN;
2927      220995594 :         else if (max_blocks > EXT_UNINIT_MAX_LEN &&
2928                :                  (flags & EXT4_GET_BLOCKS_UNINIT_EXT))
2929              0 :                 max_blocks = EXT_UNINIT_MAX_LEN;
2930                :
2931                :         /* Check if we can really insert (iblock)::(iblock+max_blocks) extent */
2932      220995594 :         newex.ee_block = cpu_to_le32(iblock);
2933      220995594 :         newex.ee_len = cpu_to_le16(max_blocks);
2934      220995594 :         err = ext4_ext_check_overlap(inode, &newex, path);
2935      219637727 :         if (err)
2936              4 :                 allocated = ext4_ext_get_actual_len(&newex);
2937                :         else
2938      219637723 :                 allocated = max_blocks;
2939                :
2940                :         /* allocate new block */
2941      219637727 :         ar.inode = inode;
2942      219637727 :         ar.goal = ext4_ext_find_goal(inode, path, iblock);
2943      219171512 :         ar.logical = iblock;
2944      219171512 :         ar.len = allocated;
2945      219171512 :         if (S_ISREG(inode->i_mode))
2946      218356737 :                 ar.flags = EXT4_MB_HINT_DATA;
2947                :         else
2948                :                 /* disable in-core preallocation for non-regular files */
2949         814775 :                 ar.flags = 0;
2950      219171512 :         newblock = ext4_mb_new_blocks(handle, &ar, &err);
2951      218641844 :         if (!newblock)
2952              10 :                 goto out2;
2953                :         ext_debug("allocate new block: goal %llu, found %llu/%lu\n",
2954                :                   ar.goal, newblock, allocated);
2955                :
2956                :         /* try to insert new extent into found leaf and return */
2957                :         ext4_ext_store_pblock(&newex, newblock);
2958      218641834 :         newex.ee_len = cpu_to_le16(ar.len);
2959      218641834 :         if (flags & EXT4_GET_BLOCKS_UNINIT_EXT)  /* Mark uninitialized */
2960                :                 ext4_ext_mark_uninitialized(&newex);
2961      218641834 :         err = ext4_ext_insert_extent(handle, inode, path, &newex);
2962      219799057 :         if (err) {
2963                :                 /* free data blocks we just allocated */
2964                :                 /* not a good idea to call discard here directly,
2965                :                  * but otherwise we'd need to call it every free() */
2966              0 :                 ext4_discard_preallocations(inode);
2967              0 :                 ext4_free_blocks(handle, inode, ext_pblock(&newex),
2968                :                                  ext4_ext_get_actual_len(&newex), 0);
2969              0 :                 goto out2;
```

```
2970                :         }
2971                :
2972                :         /* previous routine could use block we allocated */
2973      219799057 :         newblock = ext_pblock(&newex);
2974      219799057 :         allocated = ext4_ext_get_actual_len(&newex);
2975      219799067 : outnew:
2976                :         set_buffer_new(bh_result);
2977                :
2978                :         /* Cache only when it is _not_ an uninitialized extent */
2979      221387954 :         if ((flags & EXT4_GET_BLOCKS_UNINIT_EXT) == 0)
2980                :                 ext4_ext_put_in_cache(inode, iblock, allocated, newblock,
2981                :                                                     EXT4_EXT_CACHE_EXTENT);
2982      319871398 : out:
2983      319871398 :         if (allocated > max_blocks)
2984       60978600 :                 allocated = max_blocks;
2985                :         ext4_ext_show_leaf(inode, path);
2986                :         set_buffer_mapped(bh_result);
2987      319797433 :         bh_result->b_bdev = inode->i_sb->s_bdev;
2988      319797433 :         bh_result->b_blocknr = newblock;
2989     1350087509 : out2:
2990     1350087509 :         if (path) {
2991      494251946 :                 ext4_ext_drop_refs(path);
2992      493495514 :                 kfree(path);
2993                :         }
2994     1333360092 :         return err ? err : allocated;
2995                : }
2996                :
2997                : void ext4_ext_truncate(struct inode *inode)
2998        1662897 : {
2999        1662897 :         struct address_space *mapping = inode->i_mapping;
3000        1662897 :         struct super_block *sb = inode->i_sb;
3001                :         ext4_lblk_t last_block;
3002                :         handle_t *handle;
3003        1662897 :         int err = 0;
3004                :
3005                :         /*
3006                :          * probably first extent we're gonna free will be last in block
3007                :          */
3008        1662897 :         err = ext4_writepage_trans_blocks(inode);
3009        1662896 :         handle = ext4_journal_start(inode, err);
3010        1662896 :         if (IS_ERR(handle))
3011              0 :                 return;
3012                :
3013        1662896 :         if (inode->i_size & (sb->s_blocksize - 1))
3014              0 :                 ext4_block_truncate_page(handle, mapping, inode->i_size);
3015                :
3016        1662896 :         if (ext4_orphan_add(handle, inode))
3017              0 :                 goto out_stop;
3018                :
3019        1662897 :         down_write(&EXT4_I(inode)->i_data_sem);
3020                :         ext4_ext_invalidate_cache(inode);
3021                :
3022        1662897 :         ext4_discard_preallocations(inode);
3023                :
3024                :         /*
3025                :          * TODO: optimization is possible here.
3026                :          * Probably we need not scan at all,
3027                :          * because page truncation is enough.
3028                :          */
3029                :
3030                :         /* we have to know where to truncate from in crash case */
3031        1662897 :         EXT4_I(inode)->i_disksize = inode->i_size;
3032        1662897 :         ext4_mark_inode_dirty(handle, inode);
3033                :
3034        1662897 :         last_block = (inode->i_size + sb->s_blocksize - 1)
3035                :                         >> EXT4_BLOCK_SIZE_BITS(sb);
3036        1662897 :         err = ext4_ext_remove_space(inode, last_block);
3037                :
3038                :         /* In a multi-transaction truncate, we only make the final
3039                :          * transaction synchronous.
3040                :          */
```

```
3041  1662891 :         if (IS_SYNC(inode))
3042          :                 ext4_handle_sync(handle);
3043          :
3044  1662891 : out_stop:
3045  1662891 :         up_write(&EXT4_I(inode)->i_data_sem);
3046          :         /*
3047          :          * If this was a simple ftruncate() and the file will remain alive,
3048          :          * then we need to clear up the orphan record which we created above.
3049          :          * However, if this was a real unlink then we were called by
3050          :          * ext4_delete_inode(), and we allow that function to clean up the
3051          :          * orphan info for us.
3052          :          */
3053  1662892 :         if (inode->i_nlink)
3054       10 :                 ext4_orphan_del(handle, inode);
3055          :
3056  1662891 :         inode->i_mtime = inode->i_ctime = ext4_current_time(inode);
3057  1662891 :         ext4_mark_inode_dirty(handle, inode);
3058  1662895 :         ext4_journal_stop(handle);
3059          : }
3060          :
3061          : static void ext4_falloc_update_inode(struct inode *inode,
3062          :                                      int mode, loff_t new_size, int update_ctime)
3063          : {
3064          :         struct timespec now;
3065          :
3066       22 :         if (update_ctime) {
3067       18 :                 now = current_fs_time(inode->i_sb);
3068       36 :                 if (!timespec_equal(&inode->i_ctime, &now))
3069        4 :                         inode->i_ctime = now;
3070          :         }
3071          :         /*
3072          :          * Update only when preallocation was requested beyond
3073          :          * the file size.
3074          :          */
3075       22 :         if (!(mode & FALLOC_FL_KEEP_SIZE)) {
3076       10 :                 if (new_size > i_size_read(inode))
3077          :                         i_size_write(inode, new_size);
3078       10 :                 if (new_size > EXT4_I(inode)->i_disksize)
3079          :                         ext4_update_i_disksize(inode, new_size);
3080          :         }
3081          :
3082          : }
3083          :
3084          : /*
3085          :  * preallocate space for a file. This implements ext4's fallocate inode
3086          :  * operation, which gets called from sys_fallocate system call.
3087          :  * For block-mapped files, posix_fallocate should fall back to the method
3088          :  * of writing zeroes to the required new blocks (the same behavior which is
3089          :  * expected for file systems which do not support fallocate() system call).
3090          :  */
3091          : long ext4_fallocate(struct inode *inode, int mode, loff_t offset, loff_t len)
3092       22 : {
3093          :         handle_t *handle;
3094          :         ext4_lblk_t block;
3095          :         loff_t new_size;
3096          :         unsigned int max_blocks;
3097       22 :         int ret = 0;
3098       22 :         int ret2 = 0;
3099       22 :         int retries = 0;
3100          :         struct buffer_head map_bh;
3101       22 :         unsigned int credits, blkbits = inode->i_blkbits;
3102          :
3103          :         /*
3104          :          * currently supporting (pre)allocate mode for extent-based
3105          :          * files _only_
3106          :          */
3107       22 :         if (!(EXT4_I(inode)->i_flags & EXT4_EXTENTS_FL))
3108        0 :                 return -EOPNOTSUPP;
3109          :
3110          :         /* preallocation to directories is currently not supported */
3111       22 :         if (S_ISDIR(inode->i_mode))
```

```
3112                0 :                 return -ENODEV;
3113                  :
3114               22 :         block = offset >> blkbits;
3115                  :         /*
3116                  :          * We can't just convert len to max_blocks because
3117                  :          * If blocksize = 4096 offset = 3072 and len = 2048
3118                  :          */
3119               22 :         max_blocks = (EXT4_BLOCK_ALIGN(len + offset, blkbits) >> blkbits)
3120                  :                                                       - block;
3121                  :         /*
3122                  :          * credits to insert 1 extent into extent tree
3123                  :          */
3124               22 :         credits = ext4_chunk_trans_blocks(inode, max_blocks);
3125               22 :         mutex_lock(&inode->i_mutex);
3126                  : retry:
3127               44 :         while (ret >= 0 && ret < max_blocks) {
3128               22 :                 block = block + ret;
3129               22 :                 max_blocks = max_blocks - ret;
3130               44 :                 handle = ext4_journal_start(inode, credits);
3131               22 :                 if (IS_ERR(handle)) {
3132                0 :                         ret = PTR_ERR(handle);
3133                0 :                         break;
3134                  :                 }
3135               22 :                 map_bh.b_state = 0;
3136               22 :                 ret = ext4_get_blocks(handle, inode, block,
3137                  :                                       max_blocks, &map_bh,
3138                  :                                       EXT4_GET_BLOCKS_CREATE_UNINIT_EXT);
3139               22 :                 if (ret <= 0) {
3140                  : #ifdef EXT4FS_DEBUG
3141                  :                         WARN_ON(ret <= 0);
3142                  :                         printk(KERN_ERR "%s: ext4_ext_get_blocks "
3143                  :                                     "returned error inode#%lu, block=%u, "
3144                  :                                     "max_blocks=%u", __func__,
3145                  :                                     inode->i_ino, block, max_blocks);
3146                  : #endif
3147                0 :                         ext4_mark_inode_dirty(handle, inode);
3148                0 :                         ret2 = ext4_journal_stop(handle);
3149                0 :                         break;
3150                  :                 }
3151               22 :                 if ((block + ret) >= (EXT4_BLOCK_ALIGN(offset + len,
3152                  :                                                        blkbits) >> blkbits))
3153               22 :                         new_size = offset + len;
3154                  :                 else
3155                0 :                         new_size = (block + ret) << blkbits;
3156                  :
3157                  :                 ext4_falloc_update_inode(inode, mode, new_size,
3158                  :                                                       buffer_new(&map_bh));
3159               22 :                 ext4_mark_inode_dirty(handle, inode);
3160               22 :                 ret2 = ext4_journal_stop(handle);
3161               22 :                 if (ret2)
3162                0 :                         break;
3163                  :         }
3164               22 :         if (ret == -ENOSPC &&
3165                  :                         ext4_should_retry_alloc(inode->i_sb, &retries)) {
3166                0 :                 ret = 0;
3167                0 :                 goto retry;
3168                  :         }
3169               22 :         mutex_unlock(&inode->i_mutex);
3170               22 :         return ret > 0 ? ret2 : ret;
3171                  : }
3172                  :
3173                  : /*
3174                  :  * Callback function called for each extent to gather FIEMAP information.
3175                  :  */
3176                  : static int ext4_ext_fiemap_cb(struct inode *inode, struct ext4_ext_path *path,
3177                  :                               struct ext4_ext_cache *newex, struct ext4_extent *ex,
3178                  :                               void *data)
3179                0 : {
3180                0 :         struct fiemap_extent_info *fieinfo = data;
3181                0 :         unsigned char blksize_bits = inode->i_sb->s_blocksize_bits;
3182                  :         __u64   logical;
```

```
3183                :         __u64   physical;
3184                :         __u64   length;
3185              0 :         __u32   flags = 0;
3186                :         int     error;
3187                :
3188              0 :         logical =  (__u64)newex->ec_block << blksize_bits;
3189                :
3190              0 :         if (newex->ec_type == EXT4_EXT_CACHE_GAP) {
3191                :                 pgoff_t offset;
3192                :                 struct page *page;
3193              0 :                 struct buffer_head *bh = NULL;
3194                :
3195              0 :                 offset = logical >> PAGE_SHIFT;
3196              0 :                 page = find_get_page(inode->i_mapping, offset);
3197              0 :                 if (!page || !page_has_buffers(page))
3198              0 :                         return EXT_CONTINUE;
3199                :
3200              0 :                 bh = page_buffers(page);
3201                :
3202              0 :                 if (!bh)
3203              0 :                         return EXT_CONTINUE;
3204                :
3205              0 :                 if (buffer_delay(bh)) {
3206              0 :                         flags |= FIEMAP_EXTENT_DELALLOC;
3207              0 :                         page_cache_release(page);
3208                :                 } else {
3209              0 :                         page_cache_release(page);
3210              0 :                         return EXT_CONTINUE;
3211                :                 }
3212                :         }
3213                :
3214              0 :         physical = (__u64)newex->ec_start << blksize_bits;
3215              0 :         length =   (__u64)newex->ec_len << blksize_bits;
3216                :
3217              0 :         if (ex && ext4_ext_is_uninitialized(ex))
3218              0 :                 flags |= FIEMAP_EXTENT_UNWRITTEN;
3219                :
3220                :         /*
3221                :          * If this extent reaches EXT_MAX_BLOCK, it must be last.
3222                :          *
3223                :          * Or if ext4_ext_next_allocated_block is EXT_MAX_BLOCK,
3224                :          * this also indicates no more allocated blocks.
3225                :          *
3226                :          * XXX this might miss a single-block extent at EXT_MAX_BLOCK
3227                :          */
3228              0 :         if (ext4_ext_next_allocated_block(path) == EXT_MAX_BLOCK ||
3229                :             newex->ec_block + newex->ec_len - 1 == EXT_MAX_BLOCK) {
3230              0 :                 loff_t size = i_size_read(inode);
3231              0 :                 loff_t bs = EXT4_BLOCK_SIZE(inode->i_sb);
3232                :
3233              0 :                 flags |= FIEMAP_EXTENT_LAST;
3234              0 :                 if ((flags & FIEMAP_EXTENT_DELALLOC) &&
3235                :                     logical+length > size)
3236              0 :                         length = (size - logical + bs - 1) & ~(bs-1);
3237                :         }
3238                :
3239              0 :         error = fiemap_fill_next_extent(fieinfo, logical, physical,
3240                :                                         length, flags);
3241              0 :         if (error < 0)
3242              0 :                 return error;
3243              0 :         if (error == 1)
3244              0 :                 return EXT_BREAK;
3245                :
3246              0 :         return EXT_CONTINUE;
3247                : }
3248                :
3249                : /* fiemap flags we can handle specified here */
3250                : #define EXT4_FIEMAP_FLAGS       (FIEMAP_FLAG_SYNC|FIEMAP_FLAG_XATTR)
3251                :
3252                : static int ext4_xattr_fiemap(struct inode *inode,
3253                :                              struct fiemap_extent_info *fieinfo)
```

```
3254                 : {
3255               0 :         __u64 physical = 0;
3256                 :         __u64 length;
3257               0 :         __u32 flags = FIEMAP_EXTENT_LAST;
3258               0 :         int blockbits = inode->i_sb->s_blocksize_bits;
3259               0 :         int error = 0;
3260                 :
3261                 :         /* in-inode? */
3262               0 :         if (EXT4_I(inode)->i_state & EXT4_STATE_XATTR) {
3263                 :                 struct ext4_iloc iloc;
3264                 :                 int offset;     /* offset of xattr in inode */
3265                 :
3266               0 :                 error = ext4_get_inode_loc(inode, &iloc);
3267               0 :                 if (error)
3268               0 :                         return error;
3269               0 :                 physical = iloc.bh->b_blocknr << blockbits;
3270               0 :                 offset = EXT4_GOOD_OLD_INODE_SIZE +
3271                 :                                 EXT4_I(inode)->i_extra_isize;
3272               0 :                 physical += offset;
3273               0 :                 length = EXT4_SB(inode->i_sb)->s_inode_size - offset;
3274               0 :                 flags |= FIEMAP_EXTENT_DATA_INLINE;
3275                 :         } else { /* external block */
3276               0 :                 physical = EXT4_I(inode)->i_file_acl << blockbits;
3277               0 :                 length = inode->i_sb->s_blocksize;
3278                 :         }
3279                 :
3280               0 :         if (physical)
3281               0 :                 error = fiemap_fill_next_extent(fieinfo, 0, physical,
3282                 :                                                 length, flags);
3283               0 :         return (error < 0 ? error : 0);
3284                 : }
3285                 :
3286                 : int ext4_fiemap(struct inode *inode, struct fiemap_extent_info *fieinfo,
3287                 :                 __u64 start, __u64 len)
3288               0 : {
3289                 :         ext4_lblk_t start_blk;
3290                 :         ext4_lblk_t len_blks;
3291               0 :         int error = 0;
3292                 :
3293                 :         /* fallback to generic here if not in extents fmt */
3294               0 :         if (!(EXT4_I(inode)->i_flags & EXT4_EXTENTS_FL))
3295               0 :                 return generic_block_fiemap(inode, fieinfo, start, len,
3296                 :                                 ext4_get_block);
3297                 :
3298               0 :         if (fiemap_check_flags(fieinfo, EXT4_FIEMAP_FLAGS))
3299               0 :                 return -EBADR;
3300                 :
3301               0 :         if (fieinfo->fi_flags & FIEMAP_FLAG_XATTR) {
3302               0 :                 error = ext4_xattr_fiemap(inode, fieinfo);
3303                 :         } else {
3304               0 :                 start_blk = start >> inode->i_sb->s_blocksize_bits;
3305               0 :                 len_blks = len >> inode->i_sb->s_blocksize_bits;
3306                 :
3307                 :                 /*
3308                 :                  * Walk the extent tree gathering extent information.
3309                 :                  * ext4_ext_fiemap_cb will push extents back to user.
3310                 :                  */
3311               0 :                 down_read(&EXT4_I(inode)->i_data_sem);
3312               0 :                 error = ext4_ext_walk_space(inode, start_blk, len_blks,
3313                 :                                             ext4_ext_fiemap_cb, fieinfo);
3314               0 :                 up_read(&EXT4_I(inode)->i_data_sem);
3315                 :         }
3316                 :
3317               0 :         return error;
3318                 : }
3319                 :
```