LCOV - code coverage report

			Found	Hit	Coverage
Current view:	directory - fs/ext4 - ext4_extents.h (source / functions)				
Test:	kernel_2_6_31_ext4_round_3.info	**Lines:**	8	8	100.0 %
Date:	2009-10-24	**Functions:**	0	0	-

```
        1             : /*
        2             :  * Copyright (c) 2003-2006, Cluster File Systems, Inc, info@clusterfs.com
        3             :  * Written by Alex Tomas <alex@clusterfs.com>
        4             :  *
        5             :  * This program is free software; you can redistribute it and/or modify
        6             :  * it under the terms of the GNU General Public License version 2 as
        7             :  * published by the Free Software Foundation.
        8             :  *
        9             :  * This program is distributed in the hope that it will be useful,
       10             :  * but WITHOUT ANY WARRANTY; without even the implied warranty of
       11             :  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
       12             :  * GNU General Public License for more details.
       13             :  *
       14             :  * You should have received a copy of the GNU General Public Licens
       15             :  * along with this program; if not, write to the Free Software
       16             :  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-
       17             :  */
       18             :
       19             : #ifndef _EXT4_EXTENTS
       20             : #define _EXT4_EXTENTS
       21             :
       22             : #include "ext4.h"
       23             :
       24             : /*
       25             :  * With AGGRESSIVE_TEST defined, the capacity of index/leaf blocks
       26             :  * becomes very small, so index split, in-depth growing and
       27             :  * other hard changes happen much more often.
       28             :  * This is for debug purposes only.
       29             :  */
       30             : #define AGGRESSIVE_TEST_
       31             :
       32             : /*
       33             :  * With EXTENTS_STATS defined, the number of blocks and extents
       34             :  * are collected in the truncate path. They'll be shown at
       35             :  * umount time.
       36             :  */
       37             : #define EXTENTS_STATS__
       38             :
       39             : /*
       40             :  * If CHECK_BINSEARCH is defined, then the results of the binary search
       41             :  * will also be checked by linear search.
       42             :  */
       43             : #define CHECK_BINSEARCH__
       44             :
       45             : /*
       46             :  * If EXT_DEBUG is defined you can use the 'extdebug' mount option
       47             :  * to get lots of info about what's going on.
       48             :  */
       49             : #define EXT_DEBUG__
       50             : #ifdef EXT_DEBUG
       51             : #define ext_debug(a...)         printk(a)
       52             : #else
       53             : #define ext_debug(a...)
       54             : #endif
       55             :
       56             : /*
       57             :  * If EXT_STATS is defined then stats numbers are collected.
       58             :  * These number will be displayed at umount time.
       59             :  */
       60             : #define EXT_STATS_
       61             :
       62             :
       63             : /*
       64             :  * ext4_inode has i_block array (60 bytes total).
       65             :  * The first 12 bytes store ext4_extent_header;
       66             :  * the remainder stores an array of ext4_extent.
       67             :  */
       68             :
       69             : /*
       70             :  * This is the extent on-disk structure.
       71             :  * It's used at the bottom of the tree.
       72             :  */
       73             : struct ext4_extent {
       74             :         __le32  ee_block;       /* first logical block extent covers */
```

```
:         __le16  ee_len;         /* number of blocks covered by extent */
:         __le16  ee_start_hi;    /* high 16 bits of physical block */
:         __le32  ee_start_lo;    /* low 32 bits of physical block */
: };
:
: /*
:  * This is index on-disk structure.
:  * It's used at all the levels except the bottom.
:  */
: struct ext4_extent_idx {
:         __le32  ei_block;       /* index covers logical blocks from 'block' */
:         __le32  ei_leaf_lo;     /* pointer to the physical block of the next *
:                                  * level. leaf or next index could be there */
:         __le16  ei_leaf_hi;     /* high 16 bits of physical block */
:         __u16   ei_unused;
: };
:
: /*
:  * Each block (leaves and indexes), even inode-stored has header.
:  */
: struct ext4_extent_header {
:         __le16  eh_magic;       /* probably will support different formats */
:         __le16  eh_entries;     /* number of valid entries */
:         __le16  eh_max;         /* capacity of store in entries */
:         __le16  eh_depth;       /* has tree real underlying blocks? */
:         __le32  eh_generation;  /* generation of the tree */
: };
:
: #define EXT4_EXT_MAGIC          cpu_to_le16(0xf30a)
:
: /*
:  * Array of ext4_ext_path contains path to some extent.
:  * Creation/lookup routines use it for traversal/splitting/etc.
:  * Truncate uses it to simulate recursive walking.
:  */
: struct ext4_ext_path {
:         ext4_fsblk_t                    p_block;
:         __u16                           p_depth;
:         struct ext4_extent              *p_ext;
:         struct ext4_extent_idx          *p_idx;
:         struct ext4_extent_header       *p_hdr;
:         struct buffer_head              *p_bh;
: };
:
: /*
:  * structure for external API
:  */
:
: #define EXT4_EXT_CACHE_NO       0
: #define EXT4_EXT_CACHE_GAP      1
: #define EXT4_EXT_CACHE_EXTENT   2
:
: /*
:  * to be called by ext4_ext_walk_space()
:  * negative retcode - error
:  * positive retcode - signal for ext4_ext_walk_space(), see below
:  * callback must return valid extent (passed or newly created)
:  */
: typedef int (*ext_prepare_callback)(struct inode *, struct ext4_ext_path *,
:                                     struct ext4_ext_cache *,
:                                     struct ext4_extent *, void *);
:
: #define EXT_CONTINUE   0
: #define EXT_BREAK      1
: #define EXT_REPEAT     2
:
: #define EXT_MAX_BLOCK   0xffffffff
:
: /*
:  * EXT_INIT_MAX_LEN is the maximum number of blocks we can have in an
:  * initialized extent. This is 2^15 and not (2^16 - 1), since we use the
:  * MSB of ee_len field in the extent datastructure to signify if this
:  * particular extent is an initialized extent or an uninitialized (i.e.
:  * preallocated).
:  * EXT_UNINIT_MAX_LEN is the maximum number of blocks we can have in an
:  * uninitialized extent.
:  * If ee_len is <= 0x8000, it is an initialized extent. Otherwise, it is an
:  * uninitialized one. In other words, if MSB of ee_len is set, it is an
:  * uninitialized extent with only one special scenario when ee_len = 0x8000.
:  * In this case we can not have an uninitialized extent of zero length and
:  * thus we make it as a special case of initialized extent with 0x8000 length.
:  * This way we get better extent-to-group alignment for initialized extents.
:  * Hence, the maximum number of blocks we can have in an *initialized*
:  * extent is 2^15 (32768) and in an *uninitialized* extent is 2^15-1 (32767).
:  */
: #define EXT_INIT_MAX_LEN        (1UL << 15)
: #define EXT_UNINIT_MAX_LEN      (EXT_INIT_MAX_LEN - 1)
:
:
```

```
       :  #define EXT_FIRST_EXTENT(__hdr__) \
       :          ((struct ext4_extent *) (((char *) (__hdr__)) +         \
       :                                   sizeof(struct ext4_extent_header)))
       :  #define EXT_FIRST_INDEX(__hdr__) \
       :          ((struct ext4_extent_idx *) (((char *) (__hdr__)) +     \
       :                                       sizeof(struct ext4_extent_header)))
       :  #define EXT_HAS_FREE_INDEX(__path__) \
       :          (le16_to_cpu((__path__)->p_hdr->eh_entries) \
       :                                       < le16_to_cpu((__path__)->p_hdr->eh_max))
       :  #define EXT_LAST_EXTENT(__hdr__) \
       :          (EXT_FIRST_EXTENT((__hdr__)) + le16_to_cpu((__hdr__)->eh_entries) - 1)
       :  #define EXT_LAST_INDEX(__hdr__) \
       :          (EXT_FIRST_INDEX((__hdr__)) + le16_to_cpu((__hdr__)->eh_entries) - 1)
       :  #define EXT_MAX_EXTENT(__hdr__) \
       :          (EXT_FIRST_EXTENT((__hdr__)) + le16_to_cpu((__hdr__)->eh_max) - 1)
       :  #define EXT_MAX_INDEX(__hdr__) \
       :          (EXT_FIRST_INDEX((__hdr__)) + le16_to_cpu((__hdr__)->eh_max) - 1)
       :
       :  static inline struct ext4_extent_header *ext_inode_hdr(struct inode *inode)
       :  {
-1412172596 :          return (struct ext4_extent_header *) EXT4_I(inode)->i_data;
       :  }
       :
       :  static inline struct ext4_extent_header *ext_block_hdr(struct buffer_head *bh)
       :  {
 549927659 :          return (struct ext4_extent_header *) bh->b_data;
       :  }
       :
       :  static inline unsigned short ext_depth(struct inode *inode)
       :  {
-1925009276 :          return le16_to_cpu(ext_inode_hdr(inode)->eh_depth);
       :  }
       :
       :  static inline void
       :  ext4_ext_invalidate_cache(struct inode *inode)
       :  {
 232570977 :          EXT4_I(inode)->i_cached_extent.ec_type = EXT4_EXT_CACHE_NO;
       :  }
       :
       :  static inline void ext4_ext_mark_uninitialized(struct ext4_extent *ext)
       :  {
       :          /* We can not have an uninitialized extent of zero length! */
        20 :          BUG_ON((le16_to_cpu(ext->ee_len) & ~EXT_INIT_MAX_LEN) == 0);
        20 :          ext->ee_len |= cpu_to_le16(EXT_INIT_MAX_LEN);
       :  }
       :
       :  static inline int ext4_ext_is_uninitialized(struct ext4_extent *ext)
       :  {
       :          /* Extent with ee_len of 0x8000 is treated as an initialized extent */
 674241306 :          return (le16_to_cpu(ext->ee_len) > EXT_INIT_MAX_LEN);
       :  }
       :
       :  static inline int ext4_ext_get_actual_len(struct ext4_extent *ext)
       :  {
-1434573617 :          return (le16_to_cpu(ext->ee_len) <= EXT_INIT_MAX_LEN ?
       :                  le16_to_cpu(ext->ee_len) :
       :                  (le16_to_cpu(ext->ee_len) - EXT_INIT_MAX_LEN));
       :  }
       :
       :  extern int ext4_ext_calc_metadata_amount(struct inode *inode, int blocks);
       :  extern ext4_fsblk_t ext_pblock(struct ext4_extent *ex);
       :  extern ext4_fsblk_t idx_pblock(struct ext4_extent_idx *);
       :  extern void ext4_ext_store_pblock(struct ext4_extent *, ext4_fsblk_t);
       :  extern int ext4_extent_tree_init(handle_t *, struct inode *);
       :  extern int ext4_ext_calc_credits_for_single_extent(struct inode *inode,
       :                                                     int num,
       :                                                     struct ext4_ext_path *path);
       :  extern int ext4_can_extents_be_merged(struct inode *inode,
       :                                        struct ext4_extent *ex1,
       :                                        struct ext4_extent *ex2);
       :  extern int ext4_ext_try_to_merge(struct inode *inode,
       :                                   struct ext4_ext_path *path,
       :                                   struct ext4_extent *);
       :  extern unsigned int ext4_ext_check_overlap(struct inode *, struct ext4_extent *, struct ext4_ext_path *);
       :  extern int ext4_ext_insert_extent(handle_t *, struct inode *, struct ext4_ext_path *, struct ext4_extent *);
       :  extern int ext4_ext_walk_space(struct inode *, ext4_lblk_t, ext4_lblk_t,
       :                                                           ext_prepare_callback, void *);
       :  extern struct ext4_ext_path *ext4_ext_find_extent(struct inode *, ext4_lblk_t,
       :                                                    struct ext4_ext_path *);
       :  extern int ext4_ext_search_left(struct inode *, struct ext4_ext_path *,
       :                                                  ext4_lblk_t *, ext4_fsblk_t *);
       :  extern int ext4_ext_search_right(struct inode *, struct ext4_ext_path *,
       :                                                   ext4_lblk_t *, ext4_fsblk_t *);
       :  extern void ext4_ext_drop_refs(struct ext4_ext_path *);
       :  extern int ext4_ext_check_inode(struct inode *inode);
       :  #endif /* _EXT4_EXTENTS */
       :
```