LCOV - code coverage report

			Found	Hit	Coverage
Current view:	directory - fs/ext4 - ext4.h (source / functions)				
Test:	kernel_2_6_31_ext4_round_3.info	**Lines:**	47	42	89.4 %
Date:	2009-10-24	**Functions:**	3	3	100.0 %

```
                :/*
                : *  ext4.h
                : *
                : * Copyright (C) 1992, 1993, 1994, 1995
                : * Remy Card (card@masi.ibp.fr)
                : * Laboratoire MASI - Institut Blaise Pascal
                : * Universite Pierre et Marie Curie (Paris VI)
                : *
                : *  from
                : *
                : *  linux/include/linux/minix_fs.h
                : *
                : *  Copyright (C) 1991, 1992  Linus Torvalds
                : */
                :
                :#ifndef _EXT4_H
                :#define _EXT4_H
                :
                :#include <linux/types.h>
                :#include <linux/blkdev.h>
                :#include <linux/magic.h>
                :#include <linux/jbd2.h>
                :#include <linux/quota.h>
                :#include <linux/rwsem.h>
                :#include <linux/rbtree.h>
                :#include <linux/seqlock.h>
                :#include <linux/mutex.h>
                :#include <linux/timer.h>
                :#include <linux/wait.h>
                :#include <linux/blockgroup_lock.h>
                :#include <linux/percpu_counter.h>
                :
                :/*
                : * The fourth extended filesystem constants/structures
                : */
                :
                :/*
                : * Define EXT4FS_DEBUG to produce debug messages
                : */
                :#undef EXT4FS_DEBUG
                :
                :/*
                : * Debug code
                : */
                :#ifdef EXT4FS_DEBUG
                :#define ext4_debug(f, a...)                                             \
                :        do {                                                           \
                :                printk(KERN_DEBUG "EXT4-fs DEBUG (%s, %d): %s:",      \
                :                        __FILE__, __LINE__, __func__);                 \
                :                printk(KERN_DEBUG f, ## a);                            \
                :        } while (0)
                :#else
                :#define ext4_debug(f, a...)     do {} while (0)
                :#endif
                :
                :/* data type for block offset of block group */
                :typedef int ext4_grpblk_t;
                :
                :/* data type for filesystem-wide blocks number */
                :typedef unsigned long long ext4_fsblk_t;
                :
                :/* data type for file logical block number */
                :typedef __u32 ext4_lblk_t;
                :
                :/* data type for block group number */
                :typedef unsigned int ext4_group_t;
                :
                :
                :/* prefer goal again. length */
                :#define EXT4_MB_HINT_MERGE              1
                :/* blocks already reserved */
                :#define EXT4_MB_HINT_RESERVED           2
                :/* metadata is being allocated */
                :#define EXT4_MB_HINT_METADATA           4
                :/* first blocks in the file */
```

```
: #define EXT4_MB_HINT_FIRST              8
: /* search for the best chunk */
: #define EXT4_MB_HINT_BEST               16
: /* data is being allocated */
: #define EXT4_MB_HINT_DATA               32
: /* don't preallocate (for tails) */
: #define EXT4_MB_HINT_NOPREALLOC         64
: /* allocate for locality group */
: #define EXT4_MB_HINT_GROUP_ALLOC        128
: /* allocate goal blocks or none */
: #define EXT4_MB_HINT_GOAL_ONLY          256
: /* goal is meaningful */
: #define EXT4_MB_HINT_TRY_GOAL           512
: /* blocks already pre-reserved by delayed allocation */
: #define EXT4_MB_DELALLOC_RESERVED      1024
:
:
: struct ext4_allocation_request {
:         /* target inode for block we're allocating */
:         struct inode *inode;
:         /* how many blocks we want to allocate */
:         unsigned int len;
:         /* logical block in target inode */
:         ext4_lblk_t logical;
:         /* the closest logical allocated block to the left */
:         ext4_lblk_t lleft;
:         /* the closest logical allocated block to the right */
:         ext4_lblk_t lright;
:         /* phys. target (a hint) */
:         ext4_fsblk_t goal;
:         /* phys. block for the closest logical allocated block to the left */
:         ext4_fsblk_t pleft;
:         /* phys. block for the closest logical allocated block to the right */
:         ext4_fsblk_t pright;
:         /* flags. see above EXT4_MB_HINT_* */
:         unsigned int flags;
: };
:
: /*
:  * Special inodes numbers
:  */
: #define EXT4_BAD_INO             1      /* Bad blocks inode */
: #define EXT4_ROOT_INO            2      /* Root inode */
: #define EXT4_BOOT_LOADER_INO     5      /* Boot loader inode */
: #define EXT4_UNDEL_DIR_INO       6      /* Undelete directory inode */
: #define EXT4_RESIZE_INO          7      /* Reserved group descriptors inode */
: #define EXT4_JOURNAL_INO         8      /* Journal inode */
:
: /* First non-reserved inode for old ext4 filesystems */
: #define EXT4_GOOD_OLD_FIRST_INO 11
:
: /*
:  * Maximal count of links to a file
:  */
: #define EXT4_LINK_MAX           65000
:
: /*
:  * Macro-instructions used to manage several block sizes
:  */
: #define EXT4_MIN_BLOCK_SIZE             1024
: #define EXT4_MAX_BLOCK_SIZE             65536
: #define EXT4_MIN_BLOCK_LOG_SIZE         10
: #ifdef __KERNEL__
: # define EXT4_BLOCK_SIZE(s)             ((s)->s_blocksize)
: #else
: # define EXT4_BLOCK_SIZE(s)             (EXT4_MIN_BLOCK_SIZE << (s)->s_log_block_size)
: #endif
: #define EXT4_ADDR_PER_BLOCK(s)          (EXT4_BLOCK_SIZE(s) / sizeof(__u32))
: #ifdef __KERNEL__
: # define EXT4_BLOCK_SIZE_BITS(s)        ((s)->s_blocksize_bits)
: #else
: # define EXT4_BLOCK_SIZE_BITS(s)        ((s)->s_log_block_size + 10)
: #endif
: #ifdef __KERNEL__
: #define EXT4_ADDR_PER_BLOCK_BITS(s)     (EXT4_SB(s)->s_addr_per_block_bits)
: #define EXT4_INODE_SIZE(s)              (EXT4_SB(s)->s_inode_size)
: #define EXT4_FIRST_INO(s)               (EXT4_SB(s)->s_first_ino)
: #else
: #define EXT4_INODE_SIZE(s)      (((s)->s_rev_level == EXT4_GOOD_OLD_REV) ? \
:                                  EXT4_GOOD_OLD_INODE_SIZE : \
:                                  (s)->s_inode_size)
: #define EXT4_FIRST_INO(s)       (((s)->s_rev_level == EXT4_GOOD_OLD_REV) ? \
:                                  EXT4_GOOD_OLD_FIRST_INO : \
:                                  (s)->s_first_ino)
: #endif
: #define EXT4_BLOCK_ALIGN(size, blkbits)         ALIGN((size), (1 << (blkbits)))
:
: /*
:  * Structure of a blocks group descriptor
```

```
:  */
: struct ext4_group_desc
: {
:         __le32  bg_block_bitmap_lo;     /* Blocks bitmap block */
:         __le32  bg_inode_bitmap_lo;     /* Inodes bitmap block */
:         __le32  bg_inode_table_lo;      /* Inodes table block */
:         __le16  bg_free_blocks_count_lo;/* Free blocks count */
:         __le16  bg_free_inodes_count_lo;/* Free inodes count */
:         __le16  bg_used_dirs_count_lo;  /* Directories count */
:         __le16  bg_flags;               /* EXT4_BG_flags (INODE_UNINIT, etc) */
:         __u32   bg_reserved[2];         /* Likely block/inode bitmap checksum */
:         __le16  bg_itable_unused_lo;    /* Unused inodes count */
:         __le16  bg_checksum;            /* crc16(sb_uuid+group+desc) */
:         __le32  bg_block_bitmap_hi;     /* Blocks bitmap block MSB */
:         __le32  bg_inode_bitmap_hi;     /* Inodes bitmap block MSB */
:         __le32  bg_inode_table_hi;      /* Inodes table block MSB */
:         __le16  bg_free_blocks_count_hi;/* Free blocks count MSB */
:         __le16  bg_free_inodes_count_hi;/* Free inodes count MSB */
:         __le16  bg_used_dirs_count_hi;  /* Directories count MSB */
:         __le16  bg_itable_unused_hi;    /* Unused inodes count MSB */
:         __u32   bg_reserved2[3];
: };
:
: /*
:  * Structure of a flex block group info
:  */
:
: struct flex_groups {
:         atomic_t free_inodes;
:         atomic_t free_blocks;
:         atomic_t used_dirs;
: };
:
: #define EXT4_BG_INODE_UNINIT    0x0001 /* Inode table/bitmap not in use */
: #define EXT4_BG_BLOCK_UNINIT    0x0002 /* Block bitmap not in use */
: #define EXT4_BG_INODE_ZEROED    0x0004 /* On-disk itable initialized to zero */
:
: /*
:  * Macro-instructions used to manage group descriptors
:  */
: #define EXT4_MIN_DESC_SIZE              32
: #define EXT4_MIN_DESC_SIZE_64BIT        64
: #define EXT4_MAX_DESC_SIZE              EXT4_MIN_BLOCK_SIZE
: #define EXT4_DESC_SIZE(s)               (EXT4_SB(s)->s_desc_size)
: #ifdef __KERNEL__
: # define EXT4_BLOCKS_PER_GROUP(s)       (EXT4_SB(s)->s_blocks_per_group)
: # define EXT4_DESC_PER_BLOCK(s)         (EXT4_SB(s)->s_desc_per_block)
: # define EXT4_INODES_PER_GROUP(s)       (EXT4_SB(s)->s_inodes_per_group)
: # define EXT4_DESC_PER_BLOCK_BITS(s)    (EXT4_SB(s)->s_desc_per_block_bits)
: #else
: # define EXT4_BLOCKS_PER_GROUP(s)       ((s)->s_blocks_per_group)
: # define EXT4_DESC_PER_BLOCK(s)         (EXT4_BLOCK_SIZE(s) / EXT4_DESC_SIZE(s))
: # define EXT4_INODES_PER_GROUP(s)       ((s)->s_inodes_per_group)
: #endif
:
: /*
:  * Constants relative to the data blocks
:  */
: #define EXT4_NDIR_BLOCKS                12
: #define EXT4_IND_BLOCK                  EXT4_NDIR_BLOCKS
: #define EXT4_DIND_BLOCK                 (EXT4_IND_BLOCK + 1)
: #define EXT4_TIND_BLOCK                 (EXT4_DIND_BLOCK + 1)
: #define EXT4_N_BLOCKS                   (EXT4_TIND_BLOCK + 1)
:
: /*
:  * Inode flags
:  */
: #define EXT4_SECRM_FL                   0x00000001 /* Secure deletion */
: #define EXT4_UNRM_FL                    0x00000002 /* Undelete */
: #define EXT4_COMPR_FL                   0x00000004 /* Compress file */
: #define EXT4_SYNC_FL                    0x00000008 /* Synchronous updates */
: #define EXT4_IMMUTABLE_FL               0x00000010 /* Immutable file */
: #define EXT4_APPEND_FL                  0x00000020 /* writes to file may only append */
: #define EXT4_NODUMP_FL                  0x00000040 /* do not dump file */
: #define EXT4_NOATIME_FL                 0x00000080 /* do not update atime */
: /* Reserved for compression usage... */
: #define EXT4_DIRTY_FL                   0x00000100
: #define EXT4_COMPRBLK_FL                0x00000200 /* One or more compressed clusters */
: #define EXT4_NOCOMPR_FL                 0x00000400 /* Don't compress */
: #define EXT4_ECOMPR_FL                  0x00000800 /* Compression error */
: /* End compression flags --- maybe not all used */
: #define EXT4_INDEX_FL                   0x00001000 /* hash-indexed directory */
: #define EXT4_IMAGIC_FL                  0x00002000 /* AFS directory */
: #define EXT4_JOURNAL_DATA_FL            0x00004000 /* file data should be journaled */
: #define EXT4_NOTAIL_FL                  0x00008000 /* file tail should not be merged */
: #define EXT4_DIRSYNC_FL                 0x00010000 /* dirsync behaviour (directories only) */
: #define EXT4_TOPDIR_FL                  0x00020000 /* Top of directory hierarchies*/
: #define EXT4_HUGE_FILE_FL               0x00040000 /* Set to each huge file */
: #define EXT4_EXTENTS_FL                 0x00080000 /* Inode uses extents */
```

```
                : #define EXT4_EXT_MIGRATE                0x00100000 /* Inode is migrating */
                : #define EXT4_RESERVED_FL                0x80000000 /* reserved for ext4 lib */
                :
                : #define EXT4_FL_USER_VISIBLE            0x000BDFFF /* User visible flags */
                : #define EXT4_FL_USER_MODIFIABLE         0x000B80FF /* User modifiable flags */
                :
                : /* Flags that should be inherited by new inodes from their parent. */
                : #define EXT4_FL_INHERITED (EXT4_SECRM_FL | EXT4_UNRM_FL | EXT4_COMPR_FL |\
                :                           EXT4_SYNC_FL | EXT4_IMMUTABLE_FL | EXT4_APPEND_FL |\
                :                           EXT4_NODUMP_FL | EXT4_NOATIME_FL |\
                :                           EXT4_NOCOMPR_FL | EXT4_JOURNAL_DATA_FL |\
                :                           EXT4_NOTAIL_FL | EXT4_DIRSYNC_FL)
                :
                : /* Flags that are appropriate for regular files (all but dir-specific ones). */
                : #define EXT4_REG_FLMASK (~(EXT4_DIRSYNC_FL | EXT4_TOPDIR_FL))
                :
                : /* Flags that are appropriate for non-directories/regular files. */
                : #define EXT4_OTHER_FLMASK (EXT4_NODUMP_FL | EXT4_NOATIME_FL)
                :
                : /* Mask out flags that are inappropriate for the given type of inode. */
                : static inline __u32 ext4_mask_flags(umode_t mode, __u32 flags)
                : {
        9312572 :         if (S_ISDIR(mode))
         697666 :                 return flags;
        8614906 :         else if (S_ISREG(mode))
        8614906 :                 return flags & EXT4_REG_FLMASK;
                :         else
              0 :                 return flags & EXT4_OTHER_FLMASK;
                : }
                :
                : /*
                :  * Inode dynamic state flags
                :  */
                : #define EXT4_STATE_JDATA                0x00000001 /* journaled data exists */
                : #define EXT4_STATE_NEW                  0x00000002 /* inode is newly created */
                : #define EXT4_STATE_XATTR                0x00000004 /* has in-inode xattrs */
                : #define EXT4_STATE_NO_EXPAND            0x00000008 /* No space for expansion */
                : #define EXT4_STATE_DA_ALLOC_CLOSE       0x00000010 /* Alloc DA blks on close */
                :
                : /* Used to pass group descriptor data when online resize is done */
                : struct ext4_new_group_input {
                :         __u32 group;            /* Group number for this data */
                :         __u64 block_bitmap;     /* Absolute block number of block bitmap */
                :         __u64 inode_bitmap;     /* Absolute block number of inode bitmap */
                :         __u64 inode_table;      /* Absolute block number of inode table start */
                :         __u32 blocks_count;     /* Total number of blocks in this group */
                :         __u16 reserved_blocks;  /* Number of reserved blocks in this group */
                :         __u16 unused;
                : };
                :
                : /* The struct ext4_new_group_input in kernel space, with free_blocks_count */
                : struct ext4_new_group_data {
                :         __u32 group;
                :         __u64 block_bitmap;
                :         __u64 inode_bitmap;
                :         __u64 inode_table;
                :         __u32 blocks_count;
                :         __u16 reserved_blocks;
                :         __u16 unused;
                :         __u32 free_blocks_count;
                : };
                :
                : /*
                :  * Flags used by ext4_get_blocks()
                :  */
                :         /* Allocate any needed blocks and/or convert an unitialized
                :            extent to be an initialized ext4 */
                : #define EXT4_GET_BLOCKS_CREATE                  0x0001
                :         /* Request the creation of an unitialized extent */
                : #define EXT4_GET_BLOCKS_UNINIT_EXT              0x0002
                : #define EXT4_GET_BLOCKS_CREATE_UNINIT_EXT       (EXT4_GET_BLOCKS_UNINIT_EXT|\
                :                                                 EXT4_GET_BLOCKS_CREATE)
                :         /* Caller is from the delayed allocation writeout path,
                :            so set the magic i_delalloc_reserve_flag after taking the
                :            inode allocation semaphore for */
                : #define EXT4_GET_BLOCKS_DELALLOC_RESERVE        0x0004
                :         /* Call ext4_da_update_reserve_space() after successfully
                :            allocating the blocks */
                : #define EXT4_GET_BLOCKS_UPDATE_RESERVE_SPACE    0x0008
                :
                :
                : /*
                :  * ioctl commands
                :  */
                : #define EXT4_IOC_GETFLAGS               FS_IOC_GETFLAGS
                : #define EXT4_IOC_SETFLAGS               FS_IOC_SETFLAGS
                : #define EXT4_IOC_GETVERSION             _IOR('f', 3, long)
                : #define EXT4_IOC_SETVERSION             _IOW('f', 4, long)
                : #define EXT4_IOC_GETVERSION_OLD         FS_IOC_GETVERSION
```

```
343                : #define EXT4_IOC_SETVERSION_OLD         FS_IOC_SETVERSION
344                : #ifdef CONFIG_JBD2_DEBUG
345                : #define EXT4_IOC_WAIT_FOR_READONLY      _IOR('f', 99, long)
346                : #endif
347                : #define EXT4_IOC_GETRSVSZ               _IOR('f', 5, long)
348                : #define EXT4_IOC_SETRSVSZ               _IOW('f', 6, long)
349                : #define EXT4_IOC_GROUP_EXTEND           _IOW('f', 7, unsigned long)
350                : #define EXT4_IOC_GROUP_ADD              _IOW('f', 8, struct ext4_new_group_input)
351                : #define EXT4_IOC_MIGRATE                _IO('f', 9)
352                :  /* note ioctl 10 reserved for an early version of the FIEMAP ioctl */
353                :  /* note ioctl 11 reserved for filesystem-independent FIEMAP ioctl */
354                : #define EXT4_IOC_ALLOC_DA_BLKS          _IO('f', 12)
355                : #define EXT4_IOC_MOVE_EXT               _IOWR('f', 15, struct move_extent)
356                :
357                : /*
358                :  * ioctl commands in 32 bit emulation
359                :  */
360                : #define EXT4_IOC32_GETFLAGS             FS_IOC32_GETFLAGS
361                : #define EXT4_IOC32_SETFLAGS             FS_IOC32_SETFLAGS
362                : #define EXT4_IOC32_GETVERSION           _IOR('f', 3, int)
363                : #define EXT4_IOC32_SETVERSION           _IOW('f', 4, int)
364                : #define EXT4_IOC32_GETRSVSZ             _IOR('f', 5, int)
365                : #define EXT4_IOC32_SETRSVSZ             _IOW('f', 6, int)
366                : #define EXT4_IOC32_GROUP_EXTEND         _IOW('f', 7, unsigned int)
367                : #ifdef CONFIG_JBD2_DEBUG
368                : #define EXT4_IOC32_WAIT_FOR_READONLY    _IOR('f', 99, int)
369                : #endif
370                : #define EXT4_IOC32_GETVERSION_OLD       FS_IOC32_GETVERSION
371                : #define EXT4_IOC32_SETVERSION_OLD       FS_IOC32_SETVERSION
372                :
373                :
374                : /*
375                :  *  Mount options
376                :  */
377                : struct ext4_mount_options {
378                :         unsigned long s_mount_opt;
379                :         uid_t s_resuid;
380                :         gid_t s_resgid;
381                :         unsigned long s_commit_interval;
382                :         u32 s_min_batch_time, s_max_batch_time;
383                : #ifdef CONFIG_QUOTA
384                :         int s_jquota_fmt;
385                :         char *s_qf_names[MAXQUOTAS];
386                : #endif
387                : };
388                :
389                : /*
390                :  * Structure of an inode on the disk
391                :  */
392                : struct ext4_inode {
393                :         __le16  i_mode;         /* File mode */
394                :         __le16  i_uid;          /* Low 16 bits of Owner Uid */
395                :         __le32  i_size_lo;      /* Size in bytes */
396                :         __le32  i_atime;        /* Access time */
397                :         __le32  i_ctime;        /* Inode Change time */
398                :         __le32  i_mtime;        /* Modification time */
399                :         __le32  i_dtime;        /* Deletion Time */
400                :         __le16  i_gid;          /* Low 16 bits of Group Id */
401                :         __le16  i_links_count;  /* Links count */
402                :         __le32  i_blocks_lo;    /* Blocks count */
403                :         __le32  i_flags;        /* File flags */
404                :         union {
405                :                 struct {
406                :                         __le32  l_i_version;
407                :                 } linux1;
408                :                 struct {
409                :                         __u32  h_i_translator;
410                :                 } hurd1;
411                :                 struct {
412                :                         __u32  m_i_reserved1;
413                :                 } masix1;
414                :         } osd1;                         /* OS dependent 1 */
415                :         __le32  i_block[EXT4_N_BLOCKS];/* Pointers to blocks */
416                :         __le32  i_generation;   /* File version (for NFS) */
417                :         __le32  i_file_acl_lo;  /* File ACL */
418                :         __le32  i_size_high;
419                :         __le32  i_obso_faddr;   /* Obsoleted fragment address */
420                :         union {
421                :                 struct {
422                :                         __le16  l_i_blocks_high; /* were l_i_reserved1 */
423                :                         __le16  l_i_file_acl_high;
424                :                         __le16  l_i_uid_high;   /* these 2 fields */
425                :                         __le16  l_i_gid_high;   /* were reserved2[0] */
426                :                         __u32   l_i_reserved2;
427                :                 } linux2;
428                :                 struct {
429                :                         __le16  h_i_reserved1;  /* Obsoleted fragment number/size which are removed in ext4 */
430                :                         __u16   h_i_mode_high;
431                :                         __u16   h_i_uid_high;
```

```
     432                :                         __u16   h_i_gid_high;
     433                :                         __u32   h_i_author;
     434                :                 } hurd2;
     435                :                 struct {
     436                :                         __le16  h_i_reserved1;  /* Obsoleted fragment number/size which are removed in ext4 */
     437                :                         __le16  m_i_file_acl_high;
     438                :                         __u32   m_i_reserved2[2];
     439                :                 } masix2;
     440                :         } osd2;                         /* OS dependent 2 */
     441                :         __le16  i_extra_isize;
     442                :         __le16  i_pad1;
     443                :         __le32  i_ctime_extra;  /* extra Change time      (nsec << 2 | epoch) */
     444                :         __le32  i_mtime_extra;  /* extra Modification time(nsec << 2 | epoch) */
     445                :         __le32  i_atime_extra;  /* extra Access time      (nsec << 2 | epoch) */
     446                :         __le32  i_crtime;       /* File Creation time */
     447                :         __le32  i_crtime_extra; /* extra FileCreationtime (nsec << 2 | epoch) */
     448                :         __le32  i_version_hi;   /* high 32 bits for 64-bit version */
     449                : };
     450                :
     451                : struct move_extent {
     452                :         __u32 reserved;         /* should be zero */
     453                :         __u32 donor_fd;         /* donor file descriptor */
     454                :         __u64 orig_start;       /* logical start offset in block for orig */
     455                :         __u64 donor_start;      /* logical start offset in block for donor */
     456                :         __u64 len;              /* block length to be moved */
     457                :         __u64 moved_len;        /* moved block length */
     458                : };
     459                : #define MAX_DEFRAG_SIZE         ((1UL<<31) - 1)
     460                :
     461                : #define EXT4_EPOCH_BITS 2
     462                : #define EXT4_EPOCH_MASK ((1 << EXT4_EPOCH_BITS) - 1)
     463                : #define EXT4_NSEC_MASK  (~0UL << EXT4_EPOCH_BITS)
     464                :
     465                : /*
     466                :  * Extended fields will fit into an inode if the filesystem was formatted
     467                :  * with large inodes (-I 256 or larger) and there are not currently any EAs
     468                :  * consuming all of the available space. For new inodes we always reserve
     469                :  * enough space for the kernel's known extended fields, but for inodes
     470                :  * created with an old kernel this might not have been the case. None of
     471                :  * the extended inode fields is critical for correct filesystem operation.
     472                :  * This macro checks if a certain field fits in the inode. Note that
     473                :  * inode-size = GOOD_OLD_INODE_SIZE + i_extra_isize
     474                :  */
     475                : #define EXT4_FITS_IN_INODE(ext4_inode, einode, field)   \
     476                :         ((offsetof(typeof(*ext4_inode), field) +        \
     477                :           sizeof((ext4_inode)->field))                  \
     478                :         <= (EXT4_GOOD_OLD_INODE_SIZE +                  \
     479                :             (einode)->i_extra_isize))                   \
     480                :
     481                : static inline __le32 ext4_encode_extra_time(struct timespec *time)
     482                : {
      -1 483                :         return cpu_to_le32((sizeof(time->tv_sec) > 4 ?
     484                :                             time->tv_sec >> 32 : 0) |
     485                :                             ((time->tv_nsec << 2) & EXT4_NSEC_MASK));
     486                : }
     487                :
     488                : static inline void ext4_decode_extra_time(struct timespec *time, __le32 extra)
     489                : {
     490                :        if (sizeof(time->tv_sec) > 4)
     491                :                time->tv_sec |= (__u64)(le32_to_cpu(extra) & EXT4_EPOCH_MASK)
     492                :                                << 32;
15094435 493                :         time->tv_nsec = (le32_to_cpu(extra) & EXT4_NSEC_MASK) >> 2;
     494                : }
     495                :
     496                : #define EXT4_INODE_SET_XTIME(xtime, inode, raw_inode)                           \
     497                : do {                                                                           \
     498                :         (raw_inode)->xtime = cpu_to_le32((inode)->xtime.tv_sec);         \
     499                :         if (EXT4_FITS_IN_INODE(raw_inode, EXT4_I(inode), xtime ## _extra))     \
     500                :                 (raw_inode)->xtime ## _extra =                                  \
     501                :                                 ext4_encode_extra_time(&(inode)->xtime);       \
     502                : } while (0)
     503                :
     504                : #define EXT4_EINODE_SET_XTIME(xtime, einode, raw_inode)                         \
     505                : do {                                                                           \
     506                :         if (EXT4_FITS_IN_INODE(raw_inode, einode, xtime))                     \
     507                :                 (raw_inode)->xtime = cpu_to_le32((einode)->xtime.tv_sec);      \
     508                :         if (EXT4_FITS_IN_INODE(raw_inode, einode, xtime ## _extra))             \
     509                :                 (raw_inode)->xtime ## _extra =                                  \
     510                :                                 ext4_encode_extra_time(&(einode)->xtime);      \
     511                : } while (0)
     512                :
     513                : #define EXT4_INODE_GET_XTIME(xtime, inode, raw_inode)                           \
     514                : do {                                                                           \
     515                :         (inode)->xtime.tv_sec = (signed)le32_to_cpu((raw_inode)->xtime);       \
     516                :         if (EXT4_FITS_IN_INODE(raw_inode, EXT4_I(inode), xtime ## _extra))     \
     517                :                 ext4_decode_extra_time(&(inode)->xtime,                         \
     518                :                                        raw_inode->xtime ## _extra);         \
     519                : } while (0)
     520                :
```

```
521             : #define EXT4_EINODE_GET_XTIME(xtime, einode, raw_inode)                         \
522             : do {                                                                           \
523             :         if (EXT4_FITS_IN_INODE(raw_inode, einode, xtime))                     \
524             :                 (einode)->xtime.tv_sec =                                \
525             :                         (signed)le32_to_cpu((raw_inode)->xtime);        \
526             :         if (EXT4_FITS_IN_INODE(raw_inode, einode, xtime ## _extra))           \
527             :                 ext4_decode_extra_time(&(einode)->xtime,                \
528             :                                        raw_inode->xtime ## _extra);     \
529             : } while (0)
530             :
531             : #define i_disk_version osd1.linux1.l_i_version
532             :
533             : #if defined(__KERNEL__) || defined(__linux__)
534             : #define i_reserved1     osd1.linux1.l_i_reserved1
535             : #define i_file_acl_high osd2.linux2.l_i_file_acl_high
536             : #define i_blocks_high   osd2.linux2.l_i_blocks_high
537             : #define i_uid_low       i_uid
538             : #define i_gid_low       i_gid
539             : #define i_uid_high      osd2.linux2.l_i_uid_high
540             : #define i_gid_high      osd2.linux2.l_i_gid_high
541             : #define i_reserved2     osd2.linux2.l_i_reserved2
542             :
543             : #elif defined(__GNU__)
544             :
545             : #define i_translator    osd1.hurd1.h_i_translator
546             : #define i_uid_high      osd2.hurd2.h_i_uid_high
547             : #define i_gid_high      osd2.hurd2.h_i_gid_high
548             : #define i_author        osd2.hurd2.h_i_author
549             :
550             : #elif defined(__masix__)
551             :
552             : #define i_reserved1     osd1.masix1.m_i_reserved1
553             : #define i_file_acl_high osd2.masix2.m_i_file_acl_high
554             : #define i_reserved2     osd2.masix2.m_i_reserved2
555             :
556             : #endif /* defined(__KERNEL__) || defined(__linux__) */
557             :
558             : /*
559             :  * storage for cached extent
560             :  */
561             : struct ext4_ext_cache {
562             :         ext4_fsblk_t    ec_start;
563             :         ext4_lblk_t     ec_block;
564             :         __u32           ec_len; /* must be 32bit to return holes */
565             :         __u32           ec_type;
566             : };
567             :
568             : /*
569             :  * fourth extended file system inode data in memory
570             :  */
571             : struct ext4_inode_info {
572             :         __le32  i_data[15];     /* unconverted */
573             :         __u32   i_flags;
574             :         ext4_fsblk_t    i_file_acl;
575             :         __u32   i_dtime;
576             :
577             :         /*
578             :          * i_block_group is the number of the block group which contains
579             :          * this file's inode.  Constant across the lifetime of the inode,
580             :          * it is ued for making block allocation decisions - we try to
581             :          * place a file's data blocks near its inode block, and new inodes
582             :          * near to their parent directory's inode.
583             :          */
584             :         ext4_group_t    i_block_group;
585             :         __u32   i_state;                /* Dynamic state flags for ext4 */
586             :
587             :         ext4_lblk_t             i_dir_start_lookup;
588             : #ifdef CONFIG_EXT4_FS_XATTR
589             :         /*
590             :          * Extended attributes can be read independently of the main file
591             :          * data. Taking i_mutex even when reading would cause contention
592             :          * between readers of EAs and writers of regular file data, so
593             :          * instead we synchronize on xattr_sem when reading or changing
594             :          * EAs.
595             :          */
596             :         struct rw_semaphore xattr_sem;
597             : #endif
598             :
599             :         struct list_head i_orphan;      /* unlinked but open inodes */
600             :
601             :         /*
602             :          * i_disksize keeps track of what the inode size is ON DISK, not
603             :          * in memory.  During truncate, i_size is set to the new size by
604             :          * the VFS prior to calling ext4_truncate(), but the filesystem won't
605             :          * set i_disksize to 0 until the truncate is actually under way.
606             :          *
607             :          * The intent is that i_disksize always represents the blocks which
608             :          * are used by this file.  This allows recovery to restart truncate
609             :          * on orphans if we crash during truncate.  We actually write i_disksize
```

```
610                :          * into the on-disk inode when writing inodes out, instead of i_size.
611                :          *
612                :          * The only time when i_disksize and i_size may be different is when
613                :          * a truncate is in progress.  The only things which change i_disksize
614                :          * are ext4_get_block (growth) and ext4_truncate (shrinkth).
615                :          */
616                :         loff_t  i_disksize;
617                :
618                :         /*
619                :          * i_data_sem is for serialising ext4_truncate() against
620                :          * ext4_getblock().  In the 2.4 ext2 design, great chunks of inode's
621                :          * data tree are chopped off during truncate. We can't do that in
622                :          * ext4 because whenever we perform intermediate commits during
623                :          * truncate, the inode and all the metadata blocks *must* be in a
624                :          * consistent state which allows truncation of the orphans to restart
625                :          * during recovery.  Hence we must fix the get_block-vs-truncate race
626                :          * by other means, so we have i_data_sem.
627                :          */
628                :         struct rw_semaphore i_data_sem;
629                :         struct inode vfs_inode;
630                :         struct jbd2_inode jinode;
631                :
632                :         struct ext4_ext_cache i_cached_extent;
633                :         /*
634                :          * File creation time. Its function is same as that of
635                :          * struct timespec i_{a,c,m}time in the generic inode.
636                :          */
637                :         struct timespec i_crtime;
638                :
639                :         /* mballoc */
640                :         struct list_head i_prealloc_list;
641                :         spinlock_t i_prealloc_lock;
642                :
643                :         /* ialloc */
644                :         ext4_group_t    i_last_alloc_group;
645                :
646                :         /* allocation reservation info for delalloc */
647                :         unsigned int i_reserved_data_blocks;
648                :         unsigned int i_reserved_meta_blocks;
649                :         unsigned int i_allocated_meta_blocks;
650                :         unsigned short i_delalloc_reserved_flag;
651                :
652                :         /* on-disk additional length */
653                :         __u16 i_extra_isize;
654                :
655                :         spinlock_t i_block_reservation_lock;
656                : };
657                :
658                : /*
659                :  * File system states
660                :  */
661                : #define EXT4_VALID_FS                   0x0001  /* Unmounted cleanly */
662                : #define EXT4_ERROR_FS                   0x0002  /* Errors detected */
663                : #define EXT4_ORPHAN_FS                  0x0004  /* Orphans being recovered */
664                :
665                : /*
666                :  * Misc. filesystem flags
667                :  */
668                : #define EXT2_FLAGS_SIGNED_HASH          0x0001  /* Signed dirhash in use */
669                : #define EXT2_FLAGS_UNSIGNED_HASH        0x0002  /* Unsigned dirhash in use */
670                : #define EXT2_FLAGS_TEST_FILESYS         0x0004  /* to test development code */
671                :
672                : /*
673                :  * Mount flags
674                :  */
675                : #define EXT4_MOUNT_OLDALLOC             0x00002  /* Don't use the new Orlov allocator */
676                : #define EXT4_MOUNT_GRPID                0x00004 /* Create files with directory's group */
677                : #define EXT4_MOUNT_DEBUG                0x00008 /* Some debugging messages */
678                : #define EXT4_MOUNT_ERRORS_CONT          0x00010 /* Continue on errors */
679                : #define EXT4_MOUNT_ERRORS_RO            0x00020 /* Remount fs ro on errors */
680                : #define EXT4_MOUNT_ERRORS_PANIC         0x00040 /* Panic on errors */
681                : #define EXT4_MOUNT_MINIX_DF             0x00080 /* Mimics the Minix statfs */
682                : #define EXT4_MOUNT_NOLOAD               0x00100 /* Don't use existing journal*/
683                : #define EXT4_MOUNT_DATA_FLAGS           0x00C00 /* Mode for data writes: */
684                : #define EXT4_MOUNT_JOURNAL_DATA         0x00400 /* Write data to journal */
685                : #define EXT4_MOUNT_ORDERED_DATA         0x00800 /* Flush data before commit */
686                : #define EXT4_MOUNT_WRITEBACK_DATA       0x00C00 /* No data ordering */
687                : #define EXT4_MOUNT_UPDATE_JOURNAL       0x01000 /* Update the journal format */
688                : #define EXT4_MOUNT_NO_UID32             0x02000  /* Disable 32-bit UIDs */
689                : #define EXT4_MOUNT_XATTR_USER           0x04000 /* Extended user attributes */
690                : #define EXT4_MOUNT_POSIX_ACL            0x08000 /* POSIX Access Control Lists */
691                : #define EXT4_MOUNT_NO_AUTO_DA_ALLOC     0x10000 /* No auto delalloc mapping */
692                : #define EXT4_MOUNT_BARRIER              0x20000 /* Use block barriers */
693                : #define EXT4_MOUNT_NOBH                 0x40000 /* No bufferheads */
694                : #define EXT4_MOUNT_QUOTA                0x80000 /* Some quota option set */
695                : #define EXT4_MOUNT_USRQUOTA             0x100000 /* "old" user quota */
696                : #define EXT4_MOUNT_GRPQUOTA             0x200000 /* "old" group quota */
697                : #define EXT4_MOUNT_JOURNAL_CHECKSUM     0x800000 /* Journal checksums */
698                : #define EXT4_MOUNT_JOURNAL_ASYNC_COMMIT 0x1000000 /* Journal Async Commit */
```

```
699                : #define EXT4_MOUNT_I_VERSION            0x2000000 /* i_version support */
700                : #define EXT4_MOUNT_DELALLOC             0x8000000 /* Delalloc support */
701                : #define EXT4_MOUNT_DATA_ERR_ABORT       0x10000000 /* Abort on file data write */
702                : #define EXT4_MOUNT_BLOCK_VALIDITY       0x20000000 /* Block validity checking */
703                :
704                : #define clear_opt(o, opt)               o &= ~EXT4_MOUNT_##opt
705                : #define set_opt(o, opt)                 o |= EXT4_MOUNT_##opt
706                : #define test_opt(sb, opt)               (EXT4_SB(sb)->s_mount_opt & \
707                :                                         EXT4_MOUNT_##opt)
708                :
709                : #define ext4_set_bit                    ext2_set_bit
710                : #define ext4_set_bit_atomic             ext2_set_bit_atomic
711                : #define ext4_clear_bit                  ext2_clear_bit
712                : #define ext4_clear_bit_atomic           ext2_clear_bit_atomic
713                : #define ext4_test_bit                   ext2_test_bit
714                : #define ext4_find_first_zero_bit        ext2_find_first_zero_bit
715                : #define ext4_find_next_zero_bit         ext2_find_next_zero_bit
716                : #define ext4_find_next_bit              ext2_find_next_bit
717                :
718                : /*
719                :  * Maximal mount counts between two filesystem checks
720                :  */
721                : #define EXT4_DFL_MAX_MNT_COUNT          20      /* Allow 20 mounts */
722                : #define EXT4_DFL_CHECKINTERVAL          0       /* Don't use interval check */
723                :
724                : /*
725                :  * Behaviour when detecting errors
726                :  */
727                : #define EXT4_ERRORS_CONTINUE            1       /* Continue execution */
728                : #define EXT4_ERRORS_RO                  2       /* Remount fs read-only */
729                : #define EXT4_ERRORS_PANIC               3       /* Panic */
730                : #define EXT4_ERRORS_DEFAULT             EXT4_ERRORS_CONTINUE
731                :
732                : /*
733                :  * Structure of the super block
734                :  */
735                : struct ext4_super_block {
736                : /*00*/  __le32  s_inodes_count;         /* Inodes count */
737                :         __le32  s_blocks_count_lo;      /* Blocks count */
738                :         __le32  s_r_blocks_count_lo;    /* Reserved blocks count */
739                :         __le32  s_free_blocks_count_lo; /* Free blocks count */
740                : /*10*/  __le32  s_free_inodes_count;    /* Free inodes count */
741                :         __le32  s_first_data_block;     /* First Data Block */
742                :         __le32  s_log_block_size;       /* Block size */
743                :         __le32  s_obso_log_frag_size;   /* Obsoleted fragment size */
744                : /*20*/  __le32  s_blocks_per_group;     /* # Blocks per group */
745                :         __le32  s_obso_frags_per_group; /* Obsoleted fragments per group */
746                :         __le32  s_inodes_per_group;     /* # Inodes per group */
747                :         __le32  s_mtime;                /* Mount time */
748                : /*30*/  __le32  s_wtime;                /* Write time */
749                :         __le16  s_mnt_count;            /* Mount count */
750                :         __le16  s_max_mnt_count;        /* Maximal mount count */
751                :         __le16  s_magic;                /* Magic signature */
752                :         __le16  s_state;                /* File system state */
753                :         __le16  s_errors;               /* Behaviour when detecting errors */
754                :         __le16  s_minor_rev_level;      /* minor revision level */
755                : /*40*/  __le32  s_lastcheck;            /* time of last check */
756                :         __le32  s_checkinterval;        /* max. time between checks */
757                :         __le32  s_creator_os;           /* OS */
758                :         __le32  s_rev_level;            /* Revision level */
759                : /*50*/  __le16  s_def_resuid;           /* Default uid for reserved blocks */
760                :         __le16  s_def_resgid;           /* Default gid for reserved blocks */
761                :         /*
762                :          * These fields are for EXT4_DYNAMIC_REV superblocks only.
763                :          *
764                :          * Note: the difference between the compatible feature set and
765                :          * the incompatible feature set is that if there is a bit set
766                :          * in the incompatible feature set that the kernel doesn't
767                :          * know about, it should refuse to mount the filesystem.
768                :          *
769                :          * e2fsck's requirements are more strict; if it doesn't know
770                :          * about a feature in either the compatible or incompatible
771                :          * feature set, it must abort and not try to meddle with
772                :          * things it doesn't understand...
773                :          */
774                :         __le32  s_first_ino;            /* First non-reserved inode */
775                :         __le16  s_inode_size;           /* size of inode structure */
776                :         __le16  s_block_group_nr;       /* block group # of this superblock */
777                :         __le32  s_feature_compat;       /* compatible feature set */
778                : /*60*/  __le32  s_feature_incompat;     /* incompatible feature set */
779                :         __le32  s_feature_ro_compat;    /* readonly-compatible feature set */
780                : /*68*/  __u8    s_uuid[16];             /* 128-bit uuid for volume */
781                : /*78*/  char    s_volume_name[16];      /* volume name */
782                : /*88*/  char    s_last_mounted[64];     /* directory where last mounted */
783                : /*C8*/  __le32  s_algorithm_usage_bitmap; /* For compression */
784                :         /*
785                :          * Performance hints.  Directory preallocation should only
786                :          * happen if the EXT4_FEATURE_COMPAT_DIR_PREALLOC flag is on.
787                :          */
```

```
788                :         __u8    s_prealloc_blocks;      /* Nr of blocks to try to preallocate*/
789                :         __u8    s_prealloc_dir_blocks;  /* Nr to preallocate for dirs */
790                :         __le16  s_reserved_gdt_blocks;  /* Per group desc for online growth */
791                :         /*
792                :          * Journaling support valid if EXT4_FEATURE_COMPAT_HAS_JOURNAL set.
793                :          */
794                : /*D0*/  __u8    s_journal_uuid[16];     /* uuid of journal superblock */
795                : /*E0*/  __le32  s_journal_inum;         /* inode number of journal file */
796                :         __le32  s_journal_dev;          /* device number of journal file */
797                :         __le32  s_last_orphan;          /* start of list of inodes to delete */
798                :         __le32  s_hash_seed[4];         /* HTREE hash seed */
799                :         __u8    s_def_hash_version;     /* Default hash version to use */
800                :         __u8    s_reserved_char_pad;
801                :         __le16  s_desc_size;            /* size of group descriptor */
802                : /*100*/ __le32  s_default_mount_opts;
803                :         __le32  s_first_meta_bg;        /* First metablock block group */
804                :         __le32  s_mkfs_time;            /* When the filesystem was created */
805                :         __le32  s_jnl_blocks[17];       /* Backup of the journal inode */
806                :         /* 64bit support valid if EXT4_FEATURE_COMPAT_64BIT */
807                : /*150*/ __le32  s_blocks_count_hi;      /* Blocks count */
808                :         __le32  s_r_blocks_count_hi;    /* Reserved blocks count */
809                :         __le32  s_free_blocks_count_hi; /* Free blocks count */
810                :         __le16  s_min_extra_isize;      /* All inodes have at least # bytes */
811                :         __le16  s_want_extra_isize;     /* New inodes should reserve # bytes */
812                :         __le32  s_flags;                /* Miscellaneous flags */
813                :         __le16  s_raid_stride;          /* RAID stride */
814                :         __le16  s_mmp_interval;         /* # seconds to wait in MMP checking */
815                :         __le64  s_mmp_block;            /* Block for multi-mount protection */
816                :         __le32  s_raid_stripe_width;    /* blocks on all data disks (N*stride)*/
817                :         __u8    s_log_groups_per_flex;  /* FLEX_BG group size */
818                :         __u8    s_reserved_char_pad2;
819                :         __le16  s_reserved_pad;
820                :         __le64  s_kbytes_written;       /* nr of lifetime kilobytes written */
821                :         __u32   s_reserved[160];        /* Padding to the end of the block */
822                : };
823                :
824                : #ifdef __KERNEL__
825                :
826                : /*
827                :  * run-time mount flags
828                :  */
829                : #define EXT4_MF_MNTDIR_SAMPLED  0x0001
830                : #define EXT4_MF_FS_ABORTED      0x0002  /* Fatal error detected */
831                :
832                : /*
833                :  * fourth extended-fs super-block data in memory
834                :  */
835                : struct ext4_sb_info {
836                :         unsigned long s_desc_size;      /* Size of a group descriptor in bytes */
837                :         unsigned long s_inodes_per_block;/* Number of inodes per block */
838                :         unsigned long s_blocks_per_group;/* Number of blocks in a group */
839                :         unsigned long s_inodes_per_group;/* Number of inodes in a group */
840                :         unsigned long s_itb_per_group;  /* Number of inode table blocks per group */
841                :         unsigned long s_gdb_count;      /* Number of group descriptor blocks */
842                :         unsigned long s_desc_per_block; /* Number of group descriptors per block */
843                :         ext4_group_t s_groups_count;    /* Number of groups in the fs */
844                :         unsigned long s_overhead_last;  /* Last calculated overhead */
845                :         unsigned long s_blocks_last;    /* Last seen block count */
846                :         loff_t s_bitmap_maxbytes;       /* max bytes for bitmap files */
847                :         struct buffer_head * s_sbh;     /* Buffer containing the super block */
848                :         struct ext4_super_block *s_es;  /* Pointer to the super block in the buffer */
849                :         struct buffer_head **s_group_desc;
850                :         unsigned int s_mount_opt;
851                :         unsigned int s_mount_flags;
852                :         ext4_fsblk_t s_sb_block;
853                :         uid_t s_resuid;
854                :         gid_t s_resgid;
855                :         unsigned short s_mount_state;
856                :         unsigned short s_pad;
857                :         int s_addr_per_block_bits;
858                :         int s_desc_per_block_bits;
859                :         int s_inode_size;
860                :         int s_first_ino;
861                :         unsigned int s_inode_readahead_blks;
862                :         unsigned int s_inode_goal;
863                :         spinlock_t s_next_gen_lock;
864                :         u32 s_next_generation;
865                :         u32 s_hash_seed[4];
866                :         int s_def_hash_version;
867                :         int s_hash_unsigned;    /* 3 if hash should be signed, 0 if not */
868                :         struct percpu_counter s_freeblocks_counter;
869                :         struct percpu_counter s_freeinodes_counter;
870                :         struct percpu_counter s_dirs_counter;
871                :         struct percpu_counter s_dirtyblocks_counter;
872                :         struct blockgroup_lock *s_blockgroup_lock;
873                :         struct proc_dir_entry *s_proc;
874                :         struct kobject s_kobj;
875                :         struct completion s_kobj_unregister;
876                :
```

```
877                 :         /* Journaling */
878                 :         struct inode *s_journal_inode;
879                 :         struct journal_s *s_journal;
880                 :         struct list_head s_orphan;
881                 :         struct mutex s_orphan_lock;
882                 :         struct mutex s_resize_lock;
883                 :         unsigned long s_commit_interval;
884                 :         u32 s_max_batch_time;
885                 :         u32 s_min_batch_time;
886                 :         struct block_device *journal_bdev;
887                 : #ifdef CONFIG_JBD2_DEBUG
888                 :         struct timer_list turn_ro_timer;        /* For turning read-only (crash simulation) */
889                 :         wait_queue_head_t ro_wait_queue;        /* For people waiting for the fs to go read-only */
890                 : #endif
891                 : #ifdef CONFIG_QUOTA
892                 :         char *s_qf_names[MAXQUOTAS];            /* Names of quota files with journalled quota */
893                 :         int s_jquota_fmt;                       /* Format of quota to use */
894                 : #endif
895                 :         unsigned int s_want_extra_isize; /* New inodes should reserve # bytes */
896                 :         struct rb_root system_blks;
897                 :
898                 : #ifdef EXTENTS_STATS
899                 :         /* ext4 extents stats */
900                 :         unsigned long s_ext_min;
901                 :         unsigned long s_ext_max;
902                 :         unsigned long s_depth_max;
903                 :         spinlock_t s_ext_stats_lock;
904                 :         unsigned long s_ext_blocks;
905                 :         unsigned long s_ext_extents;
906                 : #endif
907                 :
908                 :         /* for buddy allocator */
909                 :         struct ext4_group_info ***s_group_info;
910                 :         struct inode *s_buddy_cache;
911                 :         long s_blocks_reserved;
912                 :         spinlock_t s_reserve_lock;
913                 :         spinlock_t s_md_lock;
914                 :         tid_t s_last_transaction;
915                 :         unsigned short *s_mb_offsets;
916                 :         unsigned int *s_mb_maxs;
917                 :
918                 :         /* tunables */
919                 :         unsigned long s_stripe;
920                 :         unsigned int s_mb_stream_request;
921                 :         unsigned int s_mb_max_to_scan;
922                 :         unsigned int s_mb_min_to_scan;
923                 :         unsigned int s_mb_stats;
924                 :         unsigned int s_mb_order2_reqs;
925                 :         unsigned int s_mb_group_prealloc;
926                 :         /* where last allocation was done - for stream allocation */
927                 :         unsigned long s_mb_last_group;
928                 :         unsigned long s_mb_last_start;
929                 :
930                 :         /* history to debug policy */
931                 :         struct ext4_mb_history *s_mb_history;
932                 :         int s_mb_history_cur;
933                 :         int s_mb_history_max;
934                 :         int s_mb_history_num;
935                 :         spinlock_t s_mb_history_lock;
936                 :         int s_mb_history_filter;
937                 :
938                 :         /* stats for buddy allocator */
939                 :         spinlock_t s_mb_pa_lock;
940                 :         atomic_t s_bal_reqs;    /* number of reqs with len > 1 */
941                 :         atomic_t s_bal_success; /* we found long enough chunks */
942                 :         atomic_t s_bal_allocated;       /* in blocks */
943                 :         atomic_t s_bal_ex_scanned;      /* total extents scanned */
944                 :         atomic_t s_bal_goals;   /* goal hits */
945                 :         atomic_t s_bal_breaks;  /* too long searches */
946                 :         atomic_t s_bal_2orders; /* 2^order hits */
947                 :         spinlock_t s_bal_lock;
948                 :         unsigned long s_mb_buddies_generated;
949                 :         unsigned long long s_mb_generation_time;
950                 :         atomic_t s_mb_lost_chunks;
951                 :         atomic_t s_mb_preallocated;
952                 :         atomic_t s_mb_discarded;
953                 :
954                 :         /* locality groups */
955                 :         struct ext4_locality_group *s_locality_groups;
956                 :
957                 :         /* for write statistics */
958                 :         unsigned long s_sectors_written_start;
959                 :         u64 s_kbytes_written;
960                 :
961                 :         unsigned int s_log_groups_per_flex;
962                 :         struct flex_groups *s_flex_groups;
963                 : };
964                 :
965                 : static inline struct ext4_sb_info *EXT4_SB(struct super_block *sb)
```

```
    966                 : {
    967              -1 :         return sb->s_fs_info;
    968                 : }
    969                 : static inline struct ext4_inode_info *EXT4_I(struct inode *inode)
    970                 : {
    971              -1 :         return container_of(inode, struct ext4_inode_info, vfs_inode);
    972                 : }
    973                 :
    974                 : static inline struct timespec ext4_current_time(struct inode *inode)
    975        11985897 : {
    976        22961373 :         return (inode->i_sb->s_time_gran < NSEC_PER_SEC) ?
    977                 :                 current_fs_time(inode->i_sb) : CURRENT_TIME_SEC;
    978                 : }
    979                 :
    980                 : static inline int ext4_valid_inum(struct super_block *sb, unsigned long ino)
    981                 : {
    982              -1 :         return ino == EXT4_ROOT_INO ||
    983                 :                 ino == EXT4_JOURNAL_INO ||
    984                 :                 ino == EXT4_RESIZE_INO ||
    985                 :                 (ino >= EXT4_FIRST_INO(sb) &&
    986                 :                  ino <= le32_to_cpu(EXT4_SB(sb)->s_es->s_inodes_count));
    987                 : }
    988                 : #else
    989                 : /* Assume that user mode programs are passing in an ext4fs superblock, not
    990                 :  * a kernel struct super_block.  This will allow us to call the feature-test
    991                 :  * macros from user land. */
    992                 : #define EXT4_SB(sb)     (sb)
    993                 : #endif
    994                 :
    995                 : #define NEXT_ORPHAN(inode) EXT4_I(inode)->i_dtime
    996                 :
    997                 : /*
    998                 :  * Codes for operating systems
    999                 :  */
   1000                 : #define EXT4_OS_LINUX           0
   1001                 : #define EXT4_OS_HURD            1
   1002                 : #define EXT4_OS_MASIX           2
   1003                 : #define EXT4_OS_FREEBSD         3
   1004                 : #define EXT4_OS_LITES           4
   1005                 :
   1006                 : /*
   1007                 :  * Revision levels
   1008                 :  */
   1009                 : #define EXT4_GOOD_OLD_REV       0       /* The good old (original) format */
   1010                 : #define EXT4_DYNAMIC_REV        1       /* V2 format w/ dynamic inode sizes */
   1011                 :
   1012                 : #define EXT4_CURRENT_REV        EXT4_GOOD_OLD_REV
   1013                 : #define EXT4_MAX_SUPP_REV       EXT4_DYNAMIC_REV
   1014                 :
   1015                 : #define EXT4_GOOD_OLD_INODE_SIZE 128
   1016                 :
   1017                 : /*
   1018                 :  * Feature set definitions
   1019                 :  */
   1020                 :
   1021                 : #define EXT4_HAS_COMPAT_FEATURE(sb,mask)                        \
   1022                 :         ((EXT4_SB(sb)->s_es->s_feature_compat & cpu_to_le32(mask)) != 0)
   1023                 : #define EXT4_HAS_RO_COMPAT_FEATURE(sb,mask)                     \
   1024                 :         ((EXT4_SB(sb)->s_es->s_feature_ro_compat & cpu_to_le32(mask)) != 0)
   1025                 : #define EXT4_HAS_INCOMPAT_FEATURE(sb,mask)                      \
   1026                 :         ((EXT4_SB(sb)->s_es->s_feature_incompat & cpu_to_le32(mask)) != 0)
   1027                 : #define EXT4_SET_COMPAT_FEATURE(sb,mask)                        \
   1028                 :         EXT4_SB(sb)->s_es->s_feature_compat |= cpu_to_le32(mask)
   1029                 : #define EXT4_SET_RO_COMPAT_FEATURE(sb,mask)                     \
   1030                 :         EXT4_SB(sb)->s_es->s_feature_ro_compat |= cpu_to_le32(mask)
   1031                 : #define EXT4_SET_INCOMPAT_FEATURE(sb,mask)                      \
   1032                 :         EXT4_SB(sb)->s_es->s_feature_incompat |= cpu_to_le32(mask)
   1033                 : #define EXT4_CLEAR_COMPAT_FEATURE(sb,mask)                      \
   1034                 :         EXT4_SB(sb)->s_es->s_feature_compat &= ~cpu_to_le32(mask)
   1035                 : #define EXT4_CLEAR_RO_COMPAT_FEATURE(sb,mask)                   \
   1036                 :         EXT4_SB(sb)->s_es->s_feature_ro_compat &= ~cpu_to_le32(mask)
   1037                 : #define EXT4_CLEAR_INCOMPAT_FEATURE(sb,mask)                    \
   1038                 :         EXT4_SB(sb)->s_es->s_feature_incompat &= ~cpu_to_le32(mask)
   1039                 :
   1040                 : #define EXT4_FEATURE_COMPAT_DIR_PREALLOC        0x0001
   1041                 : #define EXT4_FEATURE_COMPAT_IMAGIC_INODES       0x0002
   1042                 : #define EXT4_FEATURE_COMPAT_HAS_JOURNAL         0x0004
   1043                 : #define EXT4_FEATURE_COMPAT_EXT_ATTR            0x0008
   1044                 : #define EXT4_FEATURE_COMPAT_RESIZE_INODE        0x0010
   1045                 : #define EXT4_FEATURE_COMPAT_DIR_INDEX           0x0020
   1046                 :
   1047                 : #define EXT4_FEATURE_RO_COMPAT_SPARSE_SUPER     0x0001
   1048                 : #define EXT4_FEATURE_RO_COMPAT_LARGE_FILE       0x0002
   1049                 : #define EXT4_FEATURE_RO_COMPAT_BTREE_DIR        0x0004
   1050                 : #define EXT4_FEATURE_RO_COMPAT_HUGE_FILE        0x0008
   1051                 : #define EXT4_FEATURE_RO_COMPAT_GDT_CSUM         0x0010
   1052                 : #define EXT4_FEATURE_RO_COMPAT_DIR_NLINK        0x0020
   1053                 : #define EXT4_FEATURE_RO_COMPAT_EXTRA_ISIZE      0x0040
   1054                 :
```

```
1055                : #define EXT4_FEATURE_INCOMPAT_COMPRESSION       0x0001
1056                : #define EXT4_FEATURE_INCOMPAT_FILETYPE          0x0002
1057                : #define EXT4_FEATURE_INCOMPAT_RECOVER           0x0004 /* Needs recovery */
1058                : #define EXT4_FEATURE_INCOMPAT_JOURNAL_DEV       0x0008 /* Journal device */
1059                : #define EXT4_FEATURE_INCOMPAT_META_BG           0x0010
1060                : #define EXT4_FEATURE_INCOMPAT_EXTENTS           0x0040 /* extents support */
1061                : #define EXT4_FEATURE_INCOMPAT_64BIT             0x0080
1062                : #define EXT4_FEATURE_INCOMPAT_MMP               0x0100
1063                : #define EXT4_FEATURE_INCOMPAT_FLEX_BG           0x0200
1064                :
1065                : #define EXT4_FEATURE_COMPAT_SUPP        EXT2_FEATURE_COMPAT_EXT_ATTR
1066                : #define EXT4_FEATURE_INCOMPAT_SUPP      (EXT4_FEATURE_INCOMPAT_FILETYPE| \
1067                :                                         EXT4_FEATURE_INCOMPAT_RECOVER| \
1068                :                                         EXT4_FEATURE_INCOMPAT_META_BG| \
1069                :                                         EXT4_FEATURE_INCOMPAT_EXTENTS| \
1070                :                                         EXT4_FEATURE_INCOMPAT_64BIT| \
1071                :                                         EXT4_FEATURE_INCOMPAT_FLEX_BG)
1072                : #define EXT4_FEATURE_RO_COMPAT_SUPP     (EXT4_FEATURE_RO_COMPAT_SPARSE_SUPER| \
1073                :                                         EXT4_FEATURE_RO_COMPAT_LARGE_FILE| \
1074                :                                         EXT4_FEATURE_RO_COMPAT_GDT_CSUM| \
1075                :                                         EXT4_FEATURE_RO_COMPAT_DIR_NLINK | \
1076                :                                         EXT4_FEATURE_RO_COMPAT_EXTRA_ISIZE | \
1077                :                                         EXT4_FEATURE_RO_COMPAT_BTREE_DIR |\
1078                :                                         EXT4_FEATURE_RO_COMPAT_HUGE_FILE)
1079                :
1080                : /*
1081                :  * Default values for user and/or group using reserved blocks
1082                :  */
1083                : #define EXT4_DEF_RESUID         0
1084                : #define EXT4_DEF_RESGID         0
1085                :
1086                : #define EXT4_DEF_INODE_READAHEAD_BLKS   32
1087                :
1088                : /*
1089                :  * Default mount options
1090                :  */
1091                : #define EXT4_DEFM_DEBUG         0x0001
1092                : #define EXT4_DEFM_BSDGROUPS     0x0002
1093                : #define EXT4_DEFM_XATTR_USER    0x0004
1094                : #define EXT4_DEFM_ACL           0x0008
1095                : #define EXT4_DEFM_UID16         0x0010
1096                : #define EXT4_DEFM_JMODE         0x0060
1097                : #define EXT4_DEFM_JMODE_DATA    0x0020
1098                : #define EXT4_DEFM_JMODE_ORDERED 0x0040
1099                : #define EXT4_DEFM_JMODE_WBACK   0x0060
1100                :
1101                : /*
1102                :  * Default journal batch times
1103                :  */
1104                : #define EXT4_DEF_MIN_BATCH_TIME 0
1105                : #define EXT4_DEF_MAX_BATCH_TIME 15000 /* 15ms */
1106                :
1107                : /*
1108                :  * Minimum number of groups in a flexgroup before we separate out
1109                :  * directories into the first block group of a flexgroup
1110                :  */
1111                : #define EXT4_FLEX_SIZE_DIR_ALLOC_SCHEME 4
1112                :
1113                : /*
1114                :  * Structure of a directory entry
1115                :  */
1116                : #define EXT4_NAME_LEN 255
1117                :
1118                : struct ext4_dir_entry {
1119                :         __le32  inode;                  /* Inode number */
1120                :         __le16  rec_len;                /* Directory entry length */
1121                :         __le16  name_len;               /* Name length */
1122                :         char    name[EXT4_NAME_LEN];    /* File name */
1123                : };
1124                :
1125                : /*
1126                :  * The new version of the directory entry.  Since EXT4 structures are
1127                :  * stored in intel byte order, and the name_len field could never be
1128                :  * bigger than 255 chars, it's safe to reclaim the extra byte for the
1129                :  * file_type field.
1130                :  */
1131                : struct ext4_dir_entry_2 {
1132                :         __le32  inode;                  /* Inode number */
1133                :         __le16  rec_len;                /* Directory entry length */
1134                :         __u8    name_len;               /* Name length */
1135                :         __u8    file_type;
1136                :         char    name[EXT4_NAME_LEN];    /* File name */
1137                : };
1138                :
1139                : /*
1140                :  * Ext4 directory file types.  Only the low 3 bits are used.  The
1141                :  * other bits are reserved for now.
1142                :  */
1143                : #define EXT4_FT_UNKNOWN         0
```

```
                : #define EXT4_FT_REG_FILE        1
                : #define EXT4_FT_DIR             2
                : #define EXT4_FT_CHRDEV          3
                : #define EXT4_FT_BLKDEV          4
                : #define EXT4_FT_FIFO            5
                : #define EXT4_FT_SOCK            6
                : #define EXT4_FT_SYMLINK         7
                :
                : #define EXT4_FT_MAX             8
                :
                : /*
                :  * EXT4_DIR_PAD defines the directory entries boundaries
                :  *
                :  * NOTE: It must be a multiple of 4
                :  */
                : #define EXT4_DIR_PAD                    4
                : #define EXT4_DIR_ROUND                  (EXT4_DIR_PAD - 1)
                : #define EXT4_DIR_REC_LEN(name_len)      (((name_len) + 8 + EXT4_DIR_ROUND) & \
                :                                         ~EXT4_DIR_ROUND)
                : #define EXT4_MAX_REC_LEN                ((1<<16)-1)
                :
                : /*
                :  * Hash Tree Directory indexing
                :  * (c) Daniel Phillips, 2001
                :  */
                :
                : #define is_dx(dir) (EXT4_HAS_COMPAT_FEATURE(dir->i_sb, \
                :                                    EXT4_FEATURE_COMPAT_DIR_INDEX) && \
                :                           (EXT4_I(dir)->i_flags & EXT4_INDEX_FL))
                : #define EXT4_DIR_LINK_MAX(dir) (!is_dx(dir) && (dir)->i_nlink >= EXT4_LINK_MAX)
                : #define EXT4_DIR_LINK_EMPTY(dir) ((dir)->i_nlink == 2 || (dir)->i_nlink == 1)
                :
                : /* Legal values for the dx_root hash_version field: */
                :
                : #define DX_HASH_LEGACY          0
                : #define DX_HASH_HALF_MD4        1
                : #define DX_HASH_TEA             2
                : #define DX_HASH_LEGACY_UNSIGNED 3
                : #define DX_HASH_HALF_MD4_UNSIGNED       4
                : #define DX_HASH_TEA_UNSIGNED            5
                :
                : #ifdef __KERNEL__
                :
                : /* hash info structure used by the directory hash */
                : struct dx_hash_info
                : {
                :         u32             hash;
                :         u32             minor_hash;
                :         int             hash_version;
                :         u32             *seed;
                : };
                :
                : #define EXT4_HTREE_EOF  0x7fffffff
                :
                : /*
                :  * Control parameters used by ext4_htree_next_block
                :  */
                : #define HASH_NB_ALWAYS          1
                :
                :
                : /*
                :  * Describe an inode's exact location on disk and in memory
                :  */
                : struct ext4_iloc
                : {
                :         struct buffer_head *bh;
                :         unsigned long offset;
                :         ext4_group_t block_group;
                : };
                :
                : static inline struct ext4_inode *ext4_raw_inode(struct ext4_iloc *iloc)
                : {
  1455658782    :         return (struct ext4_inode *) (iloc->bh->b_data + iloc->offset);
                : }
                :
                : /*
                :  * This structure is stuffed into the struct file's private_data field
                :  * for directories.  It is where we put information so that we can do
                :  * readdir operations in hash tree order.
                :  */
                : struct dir_private_info {
                :         struct rb_root  root;
                :         struct rb_node  *curr_node;
                :         struct fname    *extra_fname;
                :         loff_t          last_pos;
                :         __u32           curr_hash;
                :         __u32           curr_minor_hash;
                :         __u32           next_hash;
                : };
```

```
                :
                : /* calculate the first block number of the group */
                : static inline ext4_fsblk_t
                : ext4_group_first_block_no(struct super_block *sb, ext4_group_t group_no)
                : {
         638301 :         return group_no * (ext4_fsblk_t)EXT4_BLOCKS_PER_GROUP(sb) +
                :                 le32_to_cpu(EXT4_SB(sb)->s_es->s_first_data_block);
                : }
                :
                : /*
                :  * Special error return code only used by dx_probe() and its callers.
                :  */
                : #define ERR_BAD_DX_DIR  -75000
                :
                : void ext4_get_group_no_and_offset(struct super_block *sb, ext4_fsblk_t blocknr,
                :                         ext4_group_t *blockgrpp, ext4_grpblk_t *offsetp);
                :
                : extern struct proc_dir_entry *ext4_proc_root;
                :
                : /*
                :  * Function prototypes
                :  */
                :
                : /*
                :  * Ok, these declarations are also in <linux/kernel.h> but none of the
                :  * ext4 source programs needs to include it so they are duplicated here.
                :  */
                : # define NORET_TYPE     /**/
                : # define ATTRIB_NORET   __attribute__((noreturn))
                : # define NORET_AND      noreturn,
                :
                : /* bitmap.c */
                : extern unsigned int ext4_count_free(struct buffer_head *, unsigned);
                :
                : /* balloc.c */
                : extern unsigned int ext4_block_group(struct super_block *sb,
                :                         ext4_fsblk_t blocknr);
                : extern ext4_grpblk_t ext4_block_group_offset(struct super_block *sb,
                :                         ext4_fsblk_t blocknr);
                : extern int ext4_bg_has_super(struct super_block *sb, ext4_group_t group);
                : extern unsigned long ext4_bg_num_gdb(struct super_block *sb,
                :                         ext4_group_t group);
                : extern ext4_fsblk_t ext4_new_meta_blocks(handle_t *handle, struct inode *inode,
                :                         ext4_fsblk_t goal, unsigned long *count, int *errp);
                : extern int ext4_claim_free_blocks(struct ext4_sb_info *sbi, s64 nblocks);
                : extern int ext4_has_free_blocks(struct ext4_sb_info *sbi, s64 nblocks);
                : extern void ext4_free_blocks(handle_t *handle, struct inode *inode,
                :                         ext4_fsblk_t block, unsigned long count, int metadata);
                : extern void ext4_add_groupblocks(handle_t *handle, struct super_block *sb,
                :                                  ext4_fsblk_t block, unsigned long count);
                : extern ext4_fsblk_t ext4_count_free_blocks(struct super_block *);
                : extern void ext4_check_blocks_bitmap(struct super_block *);
                : extern struct ext4_group_desc * ext4_get_group_desc(struct super_block * sb,
                :                                                     ext4_group_t block_group,
                :                                                     struct buffer_head ** bh);
                : extern int ext4_should_retry_alloc(struct super_block *sb, int *retries);
                : struct buffer_head *ext4_read_block_bitmap(struct super_block *sb,
                :                                            ext4_group_t block_group);
                : extern unsigned ext4_init_block_bitmap(struct super_block *sb,
                :                                        struct buffer_head *bh,
                :                                        ext4_group_t group,
                :                                        struct ext4_group_desc *desc);
                : #define ext4_free_blocks_after_init(sb, group, desc)                    \
                :                 ext4_init_block_bitmap(sb, NULL, group, desc)
                :
                : /* dir.c */
                : extern int ext4_check_dir_entry(const char *, struct inode *,
                :                                 struct ext4_dir_entry_2 *,
                :                                 struct buffer_head *, unsigned int);
                : extern int ext4_htree_store_dirent(struct file *dir_file, __u32 hash,
                :                                    __u32 minor_hash,
                :                                    struct ext4_dir_entry_2 *dirent);
                : extern void ext4_htree_free_dir_info(struct dir_private_info *p);
                :
                : /* fsync.c */
                : extern int ext4_sync_file(struct file *, struct dentry *, int);
                :
                : /* hash.c */
                : extern int ext4fs_dirhash(const char *name, int len, struct
                :                           dx_hash_info *hinfo);
                :
                : /* ialloc.c */
                : extern struct inode *ext4_new_inode(handle_t *, struct inode *, int,
                :                                     const struct qstr *qstr, __u32 goal);
                : extern void ext4_free_inode(handle_t *, struct inode *);
                : extern struct inode * ext4_orphan_get(struct super_block *, unsigned long);
                : extern unsigned long ext4_count_free_inodes(struct super_block *);
                : extern unsigned long ext4_count_dirs(struct super_block *);
                : extern void ext4_check_inodes_bitmap(struct super_block *);
```

```
: extern unsigned ext4_init_inode_bitmap(struct super_block *sb,
:                                        struct buffer_head *bh,
:                                        ext4_group_t group,
:                                        struct ext4_group_desc *desc);
: extern void mark_bitmap_end(int start_bit, int end_bit, char *bitmap);
:
: /* mballoc.c */
: extern long ext4_mb_stats;
: extern long ext4_mb_max_to_scan;
: extern int ext4_mb_init(struct super_block *, int);
: extern int ext4_mb_release(struct super_block *);
: extern ext4_fsblk_t ext4_mb_new_blocks(handle_t *,
:                                       struct ext4_allocation_request *, int *);
: extern int ext4_mb_reserve_blocks(struct super_block *, int);
: extern void ext4_discard_preallocations(struct inode *);
: extern int __init init_ext4_mballoc(void);
: extern void exit_ext4_mballoc(void);
: extern void ext4_mb_free_blocks(handle_t *, struct inode *,
:                 ext4_fsblk_t, unsigned long, int, unsigned long *);
: extern int ext4_mb_add_groupinfo(struct super_block *sb,
:                 ext4_group_t i, struct ext4_group_desc *desc);
: extern void ext4_mb_update_group_info(struct ext4_group_info *grp,
:                 ext4_grpblk_t add);
: extern int ext4_mb_get_buddy_cache_lock(struct super_block *, ext4_group_t);
: extern void ext4_mb_put_buddy_cache_lock(struct super_block *,
:                                                 ext4_group_t, int);
: /* inode.c */
: int ext4_forget(handle_t *handle, int is_metadata, struct inode *inode,
:                 struct buffer_head *bh, ext4_fsblk_t blocknr);
: struct buffer_head *ext4_getblk(handle_t *, struct inode *,
:                                                 ext4_lblk_t, int, int *);
: struct buffer_head *ext4_bread(handle_t *, struct inode *,
:                                                 ext4_lblk_t, int, int *);
: int ext4_get_block(struct inode *inode, sector_t iblock,
:                                       struct buffer_head *bh_result, int create);
:
: extern struct inode *ext4_iget(struct super_block *, unsigned long);
: extern int  ext4_write_inode(struct inode *, int);
: extern int  ext4_setattr(struct dentry *, struct iattr *);
: extern int  ext4_getattr(struct vfsmount *mnt, struct dentry *dentry,
:                                       struct kstat *stat);
: extern void ext4_delete_inode(struct inode *);
: extern int  ext4_sync_inode(handle_t *, struct inode *);
: extern void ext4_dirty_inode(struct inode *);
: extern int ext4_change_inode_journal_flag(struct inode *, int);
: extern int ext4_get_inode_loc(struct inode *, struct ext4_iloc *);
: extern int ext4_can_truncate(struct inode *inode);
: extern void ext4_truncate(struct inode *);
: extern void ext4_set_inode_flags(struct inode *);
: extern void ext4_get_inode_flags(struct ext4_inode_info *);
: extern int ext4_alloc_da_blocks(struct inode *inode);
: extern void ext4_set_aops(struct inode *inode);
: extern int ext4_writepage_trans_blocks(struct inode *);
: extern int ext4_meta_trans_blocks(struct inode *, int nrblocks, int idxblocks);
: extern int ext4_chunk_trans_blocks(struct inode *, int nrblocks);
: extern int ext4_block_truncate_page(handle_t *handle,
:                 struct address_space *mapping, loff_t from);
: extern int ext4_page_mkwrite(struct vm_area_struct *vma, struct vm_fault *vmf);
: extern qsize_t ext4_get_reserved_space(struct inode *inode);
:
: /* ioctl.c */
: extern long ext4_ioctl(struct file *, unsigned int, unsigned long);
: extern long ext4_compat_ioctl(struct file *, unsigned int, unsigned long);
:
: /* migrate.c */
: extern int ext4_ext_migrate(struct inode *);
:
: /* namei.c */
: extern unsigned int ext4_rec_len_from_disk(__le16 dlen, unsigned blocksize);
: extern __le16 ext4_rec_len_to_disk(unsigned len, unsigned blocksize);
: extern int ext4_orphan_add(handle_t *, struct inode *);
: extern int ext4_orphan_del(handle_t *, struct inode *);
: extern int ext4_htree_fill_tree(struct file *dir_file, __u32 start_hash,
:                                 __u32 start_minor_hash, __u32 *next_hash);
:
: /* resize.c */
: extern int ext4_group_add(struct super_block *sb,
:                                       struct ext4_new_group_data *input);
: extern int ext4_group_extend(struct super_block *sb,
:                                       struct ext4_super_block *es,
:                                       ext4_fsblk_t n_blocks_count);
:
: /* super.c */
: extern void ext4_error(struct super_block *, const char *, const char *, ...)
:         __attribute__ ((format (printf, 3, 4)));
: extern void __ext4_std_error(struct super_block *, const char *, int);
: extern void ext4_abort(struct super_block *, const char *, const char *, ...)
:         __attribute__ ((format (printf, 3, 4)));
: extern void ext4_warning(struct super_block *, const char *, const char *, ...)
```

```
                :         __attribute__ ((format (printf, 3, 4)));
                : extern void ext4_msg(struct super_block *, const char *, const char *, ...)
                :         __attribute__ ((format (printf, 3, 4)));
                : extern void ext4_grp_locked_error(struct super_block *, ext4_group_t,
                :                                   const char *, const char *, ...)
                :         __attribute__ ((format (printf, 4, 5)));
                : extern void ext4_update_dynamic_rev(struct super_block *sb);
                : extern int ext4_update_compat_feature(handle_t *handle, struct super_block *sb,
                :                                       __u32 compat);
                : extern int ext4_update_rocompat_feature(handle_t *handle,
                :                                         struct super_block *sb, __u32 rocompat);
                : extern int ext4_update_incompat_feature(handle_t *handle,
                :                                         struct super_block *sb, __u32 incompat);
                : extern ext4_fsblk_t ext4_block_bitmap(struct super_block *sb,
                :                                       struct ext4_group_desc *bg);
                : extern ext4_fsblk_t ext4_inode_bitmap(struct super_block *sb,
                :                                       struct ext4_group_desc *bg);
                : extern ext4_fsblk_t ext4_inode_table(struct super_block *sb,
                :                                      struct ext4_group_desc *bg);
                : extern __u32 ext4_free_blks_count(struct super_block *sb,
                :                                   struct ext4_group_desc *bg);
                : extern __u32 ext4_free_inodes_count(struct super_block *sb,
                :                                     struct ext4_group_desc *bg);
                : extern __u32 ext4_used_dirs_count(struct super_block *sb,
                :                                   struct ext4_group_desc *bg);
                : extern __u32 ext4_itable_unused_count(struct super_block *sb,
                :                                       struct ext4_group_desc *bg);
                : extern void ext4_block_bitmap_set(struct super_block *sb,
                :                                   struct ext4_group_desc *bg, ext4_fsblk_t blk);
                : extern void ext4_inode_bitmap_set(struct super_block *sb,
                :                                   struct ext4_group_desc *bg, ext4_fsblk_t blk);
                : extern void ext4_inode_table_set(struct super_block *sb,
                :                                  struct ext4_group_desc *bg, ext4_fsblk_t blk);
                : extern void ext4_free_blks_set(struct super_block *sb,
                :                                struct ext4_group_desc *bg, __u32 count);
                : extern void ext4_free_inodes_set(struct super_block *sb,
                :                                  struct ext4_group_desc *bg, __u32 count);
                : extern void ext4_used_dirs_set(struct super_block *sb,
                :                                struct ext4_group_desc *bg, __u32 count);
                : extern void ext4_itable_unused_set(struct super_block *sb,
                :                                    struct ext4_group_desc *bg, __u32 count);
                : extern __le16 ext4_group_desc_csum(struct ext4_sb_info *sbi, __u32 group,
                :                                    struct ext4_group_desc *gdp);
                : extern int ext4_group_desc_csum_verify(struct ext4_sb_info *sbi, __u32 group,
                :                                        struct ext4_group_desc *gdp);
                :
                : static inline ext4_fsblk_t ext4_blocks_count(struct ext4_super_block *es)
                : {
     1581299269 :         return ((ext4_fsblk_t)le32_to_cpu(es->s_blocks_count_hi) << 32) |
                :                le32_to_cpu(es->s_blocks_count_lo);
                : }
                :
                : static inline ext4_fsblk_t ext4_r_blocks_count(struct ext4_super_block *es)
                : {
      995426239 :         return ((ext4_fsblk_t)le32_to_cpu(es->s_r_blocks_count_hi) << 32) |
                :                le32_to_cpu(es->s_r_blocks_count_lo);
                : }
                :
                : static inline ext4_fsblk_t ext4_free_blocks_count(struct ext4_super_block *es)
                : {
                :         return ((ext4_fsblk_t)le32_to_cpu(es->s_free_blocks_count_hi) << 32) |
                :                le32_to_cpu(es->s_free_blocks_count_lo);
                : }
                :
                : static inline void ext4_blocks_count_set(struct ext4_super_block *es,
                :                                          ext4_fsblk_t blk)
                : {
              0 :         es->s_blocks_count_lo = cpu_to_le32((u32)blk);
              0 :         es->s_blocks_count_hi = cpu_to_le32(blk >> 32);
                : }
                :
                : static inline void ext4_free_blocks_count_set(struct ext4_super_block *es,
                :                                               ext4_fsblk_t blk)
                : {
          76742 :         es->s_free_blocks_count_lo = cpu_to_le32((u32)blk);
          76742 :         es->s_free_blocks_count_hi = cpu_to_le32(blk >> 32);
                : }
                :
                : static inline void ext4_r_blocks_count_set(struct ext4_super_block *es,
                :                                            ext4_fsblk_t blk)
                : {
              0 :         es->s_r_blocks_count_lo = cpu_to_le32((u32)blk);
              0 :         es->s_r_blocks_count_hi = cpu_to_le32(blk >> 32);
                : }
                :
                : static inline loff_t ext4_isize(struct ext4_inode *raw_inode)
                : {
        3773613 :         if (S_ISREG(le16_to_cpu(raw_inode->i_mode)))
        3773488 :                 return ((loff_t)le32_to_cpu(raw_inode->i_size_high) << 32) |
```

```
    1500                :                         le32_to_cpu(raw_inode->i_size_lo);
    1501                :         else
    1502              125 :                 return (loff_t) le32_to_cpu(raw_inode->i_size_lo);
    1503                : }
    1504                :
    1505                : static inline void ext4_isize_set(struct ext4_inode *raw_inode, loff_t i_size)
    1506                : {
    1507      1430288015 :         raw_inode->i_size_lo = cpu_to_le32(i_size);
    1508      1430288015 :         raw_inode->i_size_high = cpu_to_le32(i_size >> 32);
    1509                : }
    1510                :
    1511                : static inline
    1512                : struct ext4_group_info *ext4_get_group_info(struct super_block *sb,
    1513                :                                            ext4_group_t group)
    1514                : {
    1515                :          struct ext4_group_info ***grp_info;
    1516                :          long indexv, indexh;
    1517               -1 :          grp_info = EXT4_SB(sb)->s_group_info;
    1518               -1 :          indexv = group >> (EXT4_DESC_PER_BLOCK_BITS(sb));
    1519               -1 :          indexh = group & ((EXT4_DESC_PER_BLOCK(sb)) - 1);
    1520               -1 :          return grp_info[indexv][indexh];
    1521                : }
    1522                :
    1523                : /*
    1524                :  * Reading s_groups_count requires using smp_rmb() afterwards.  See
    1525                :  * the locking protocol documented in the comments of ext4_group_add()
    1526                :  * in resize.c
    1527                :  */
    1528                : static inline ext4_group_t ext4_get_groups_count(struct super_block *sb)
    1529                : {
    1530               -1 :         ext4_group_t    ngroups = EXT4_SB(sb)->s_groups_count;
    1531                :
    1532               -1 :         smp_rmb();
    1533               -1 :         return ngroups;
    1534                : }
    1535                :
    1536                : static inline ext4_group_t ext4_flex_group(struct ext4_sb_info *sbi,
    1537                :                                            ext4_group_t block_group)
    1538                : {
    1539       239388304 :         return block_group >> sbi->s_log_groups_per_flex;
    1540                : }
    1541                :
    1542                : static inline unsigned int ext4_flex_bg_size(struct ext4_sb_info *sbi)
    1543                : {
    1544               -1 :         return 1 << sbi->s_log_groups_per_flex;
    1545                : }
    1546                :
    1547                : #define ext4_std_error(sb, errno)                               \
    1548                : do {                                                            \
    1549                :         if ((errno))                                           \
    1550                :                 __ext4_std_error((sb), __func__, (errno));      \
    1551                : } while (0)
    1552                :
    1553                : #ifdef CONFIG_SMP
    1554                : /* Each CPU can accumulate percpu_counter_batch blocks in their local
    1555                :  * counters. So we need to make sure we have free blocks more
    1556                :  * than percpu_counter_batch  * nr_cpu_ids. Also add a window of 4 times.
    1557                :  */
    1558                : #define EXT4_FREEBLOCKS_WATERMARK (4 * (percpu_counter_batch * nr_cpu_ids))
    1559                : #else
    1560                : #define EXT4_FREEBLOCKS_WATERMARK 0
    1561                : #endif
    1562                :
    1563                : static inline void ext4_update_i_disksize(struct inode *inode, loff_t newsize)
    1564       219132135 : {
    1565                :         /*
    1566                :          * XXX: replace with spinlock if seen contended -bzzz
    1567                :          */
    1568       219132145 :         down_write(&EXT4_I(inode)->i_data_sem);
    1569       221898461 :         if (newsize > EXT4_I(inode)->i_disksize)
    1570       221940298 :                 EXT4_I(inode)->i_disksize = newsize;
    1571       221898461 :         up_write(&EXT4_I(inode)->i_data_sem);
    1572                :         return ;
    1573                : }
    1574                :
    1575                : struct ext4_group_info {
    1576                :         unsigned long   bb_state;
    1577                :         struct rb_root  bb_free_root;
    1578                :         unsigned short  bb_first_free;
    1579                :         unsigned short  bb_free;
    1580                :         unsigned short  bb_fragments;
    1581                :         struct          list_head bb_prealloc_list;
    1582                : #ifdef DOUBLE_CHECK
    1583                :         void            *bb_bitmap;
    1584                : #endif
    1585                :         struct rw_semaphore alloc_sem;
    1586                :         unsigned short  bb_counters[];
    1587                : };
    1588                :
```

```
      1589                : #define EXT4_GROUP_INFO_NEED_INIT_BIT   0
      1590                :
      1591                : #define EXT4_MB_GRP_NEED_INIT(grp)      \
      1592                :         (test_bit(EXT4_GROUP_INFO_NEED_INIT_BIT, &((grp)->bb_state)))
      1593                :
      1594                : static inline spinlock_t *ext4_group_lock_ptr(struct super_block *sb,
      1595                :                                               ext4_group_t group)
      1596                : {
-2145303854 (hot)       :         return bgl_lock_ptr(EXT4_SB(sb)->s_blockgroup_lock, group);
      1598                : }
      1599                :
      1600                : static inline void ext4_lock_group(struct super_block *sb, ext4_group_t group)
      1601                : {
 290846926 (hot)        :         spin_lock(ext4_group_lock_ptr(sb, group));
      1603                : }
      1604                :
      1605                : static inline void ext4_unlock_group(struct super_block *sb,
      1606                :                                      ext4_group_t group)
      1607                : {
 291119069 (hot)        :         spin_unlock(ext4_group_lock_ptr(sb, group));
      1609                : }
      1610                :
      1611                : /*
      1612                :  * Inodes and files operations
      1613                :  */
      1614                :
      1615                : /* dir.c */
      1616                : extern const struct file_operations ext4_dir_operations;
      1617                :
      1618                : /* file.c */
      1619                : extern const struct inode_operations ext4_file_inode_operations;
      1620                : extern const struct file_operations ext4_file_operations;
      1621                :
      1622                : /* namei.c */
      1623                : extern const struct inode_operations ext4_dir_inode_operations;
      1624                : extern const struct inode_operations ext4_special_inode_operations;
      1625                : extern struct dentry *ext4_get_parent(struct dentry *child);
      1626                :
      1627                : /* symlink.c */
      1628                : extern const struct inode_operations ext4_symlink_inode_operations;
      1629                : extern const struct inode_operations ext4_fast_symlink_inode_operations;
      1630                :
      1631                : /* block_validity */
      1632                : extern void ext4_release_system_zone(struct super_block *sb);
      1633                : extern int ext4_setup_system_zone(struct super_block *sb);
      1634                : extern int __init init_ext4_system_zone(void);
      1635                : extern void exit_ext4_system_zone(void);
      1636                : extern int ext4_data_block_valid(struct ext4_sb_info *sbi,
      1637                :                                  ext4_fsblk_t start_blk,
      1638                :                                  unsigned int count);
      1639                :
      1640                : /* extents.c */
      1641                : extern int ext4_ext_tree_init(handle_t *handle, struct inode *);
      1642                : extern int ext4_ext_writepage_trans_blocks(struct inode *, int);
      1643                : extern int ext4_ext_index_trans_blocks(struct inode *inode, int nrblocks,
      1644                :                                        int chunk);
      1645                : extern int ext4_ext_get_blocks(handle_t *handle, struct inode *inode,
      1646                :                                ext4_lblk_t iblock, unsigned int max_blocks,
      1647                :                                struct buffer_head *bh_result, int flags);
      1648                : extern void ext4_ext_truncate(struct inode *);
      1649                : extern void ext4_ext_init(struct super_block *);
      1650                : extern void ext4_ext_release(struct super_block *);
      1651                : extern long ext4_fallocate(struct inode *inode, int mode, loff_t offset,
      1652                :                            loff_t len);
      1653                : extern int ext4_get_blocks(handle_t *handle, struct inode *inode,
      1654                :                            sector_t block, unsigned int max_blocks,
      1655                :                            struct buffer_head *bh, int flags);
      1656                : extern int ext4_fiemap(struct inode *inode, struct fiemap_extent_info *fieinfo,
      1657                :                        __u64 start, __u64 len);
      1658                : /* move_extent.c */
      1659                : extern int ext4_move_extents(struct file *o_filp, struct file *d_filp,
      1660                :                              __u64 start_orig, __u64 start_donor,
      1661                :                              __u64 len, __u64 *moved_len);
      1662                :
      1663                :
      1664                : /*
      1665                :  * Add new method to test wether block and inode bitmaps are properly
      1666                :  * initialized. With uninit_bg reading the block from disk is not enough
      1667                :  * to mark the bitmap uptodate. We need to also zero-out the bitmap
      1668                :  */
      1669                : #define BH_BITMAP_UPTODATE BH_JBDPrivateStart
      1670                :
      1671                : static inline int bitmap_uptodate(struct buffer_head *bh)
 243415131 (hot)        : {
 486104974 (hot)        :         return (buffer_uptodate(bh) &&
      1674                :                         test_bit(BH_BITMAP_UPTODATE, &(bh)->b_state));
      1675                : }
      1676                : static inline void set_bitmap_uptodate(struct buffer_head *bh)
      1677                : {
```

```
    1678          316360 :          set_bit(BH_BITMAP_UPTODATE, &(bh)->b_state);
    1679                 : }
    1680                 :
    1681                 : #endif  /* __KERNEL__ */
    1682                 :
    1683                 : #endif  /* _EXT4_H */
```