LCOV - code coverage report

Current view: directory - fs/ext4 - file.c (source / functions)		Found	Hit	Coverag
Test: kernel_2_6_31_ext4_round_3.info	Lines:	56	43	76.8
Date: 2009-10-24	Functions:	4	4	100.0

```
                :  /*
                :   *  linux/fs/ext4/file.c
                :   *
                :   * 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/fs/minix/file.c
                :   *
                :   *  Copyright (C) 1991, 1992  Linus Torvalds
                :   *
                :   *  ext4 fs regular file handling primitives
                :   *
                :   *  64-bit file support on 64-bit platforms by Jakub Jelinek
                :   *      (jj@sunsite.ms.mff.cuni.cz)
                :   */
                :
                :  #include <linux/time.h>
                :  #include <linux/fs.h>
                :  #include <linux/jbd2.h>
                :  #include <linux/mount.h>
                :  #include <linux/path.h>
                :  #include "ext4.h"
                :  #include "ext4_jbd2.h"
                :  #include "xattr.h"
                :  #include "acl.h"
                :
                :  /*
                :   * Called when an inode is released. Note that this is different
                :   * from ext4_file_open: open gets called at every open, but release
                :   * gets called only when /all/ the files are closed.
                :   */
                :  static int ext4_release_file(struct inode *inode, struct file *filp)
         48975276 :  {
         48975276 :          if (EXT4_I(inode)->i_state & EXT4_STATE_DA_ALLOC_CLOSE) {
                0 :                  ext4_alloc_da_blocks(inode);
                0 :                  EXT4_I(inode)->i_state &= ~EXT4_STATE_DA_ALLOC_CLOSE;
                :          }
                :          /* if we are the last writer on the inode, drop the block reservation */
        102474257 :          if ((filp->f_mode & FMODE_WRITE) &&
                :                          (atomic_read(&inode->i_writecount) == 1) &&
                :                          !EXT4_I(inode)->i_reserved_data_blocks)
                :          {
         11920808 :                  down_write(&EXT4_I(inode)->i_data_sem);
         11864707 :                  ext4_discard_preallocations(inode);
         11902402 :                  up_write(&EXT4_I(inode)->i_data_sem);
                :          }
        147433956 :          if (is_dx(inode) && filp->private_data)
                0 :                  ext4_htree_free_dir_info(filp->private_data);
                :
         49144652 :          return 0;
                :  }
                :
                :  static ssize_t
```

```
      58                :  ext4_file_write(struct kiocb *iocb, const struct iovec *iov,
      59                :                  unsigned long nr_segs, loff_t pos)
      60      564349814 :  {
      61      564349814 :          struct file *file = iocb->ki_filp;
      62      564349814 :          struct inode *inode = file->f_path.dentry->d_inode;
      63                :          ssize_t ret;
      64                :          int err;
      65                :
      66                :          /*
      67                :           * If we have encountered a bitmap-format file, the size limit
      68                :           * is smaller than s_maxbytes, which is for extent-mapped files.
      69                :           */
      70                :
      71      564349814 :          if (!(EXT4_I(inode)->i_flags & EXT4_EXTENTS_FL)) {
      72        1411588 :                  struct ext4_sb_info *sbi = EXT4_SB(inode->i_sb);
      73         705794 :                  size_t length = iov_length(iov, nr_segs);
      74                :
      75         705794 :                  if (pos > sbi->s_bitmap_maxbytes)
      76              0 :                          return -EFBIG;
      77                :
      78         705794 :                  if (pos + length > sbi->s_bitmap_maxbytes) {
      79              0 :                          nr_segs = iov_shorten((struct iovec *)iov, nr_segs,
      80                :                                                sbi->s_bitmap_maxbytes - pos);
      81                :                  }
      82                :          }
      83                :
      84      564349814 :          ret = generic_file_aio_write(iocb, iov, nr_segs, pos);
      85                :          /*
      86                :           * Skip flushing if there was an error, or if nothing was written.
      87                :           */
      88      564296702 :          if (ret <= 0)
      89              2 :                  return ret;
      90                :
      91                :          /*
      92                :           * If the inode is IS_SYNC, or is O_SYNC and we are doing data
      93                :           * journalling then we need to make sure that we force the transaction
      94                :           * to disk to keep all metadata uptodate synchronously.
      95                :           */
      96      564296700 :          if (file->f_flags & O_SYNC) {
      97                :                  /*
      98                :                   * If we are non-data-journaled, then the dirty data has
      99                :                   * already been flushed to backing store by generic_osync_inode,
     100                :                   * and the inode has been flushed too if there have been any
     101                :                   * modifications other than mere timestamp updates.
     102                :                   *
     103                :                   * Open question --- do we care about flushing timestamps too
     104                :                   * if the inode is IS_SYNC?
     105                :                   */
     106              0 :                  if (!ext4_should_journal_data(inode))
     107              0 :                          return ret;
     108                :
     109                :                  goto force_commit;
     110                :          }
     111                :
     112                :          /*
     113                :           * So we know that there has been no forced data flush.  If the inode
     114                :           * is marked IS_SYNC, we need to force one ourselves.
     115                :           */
     116      564296700 :          if (!IS_SYNC(inode))
     117      564319484 :                  return ret;
     118                :
     119                :          /*
     120                :           * Open question #2 --- should we force data to disk here too?  If we
     121                :           * don't, the only impact is that data=writeback filesystems won't
     122                :           * flush data to disk automatically on IS_SYNC, only metadata (but
     123                :           * historically, that is what ext2 has done.)
     124                :           */
     125                :
     126              0 :  force_commit:
     127              0 :          err = ext4_force_commit(inode->i_sb);
     128              0 :          if (err)
```

```
   129                0 :                 return err;
   130                0 :         return ret;
   131                  : }
   132                  :
   133                  : static struct vm_operations_struct ext4_file_vm_ops = {
   134                  :         .fault          = filemap_fault,
   135                  :         .page_mkwrite   = ext4_page_mkwrite,
   136                  : };
   137                  :
   138                  : static int ext4_file_mmap(struct file *file, struct vm_area_struct *vma)
   139                2 : {
   140                2 :         struct address_space *mapping = file->f_mapping;
   141                  :
   142                2 :         if (!mapping->a_ops->readpage)
   143                0 :                 return -ENOEXEC;
   144                  :         file_accessed(file);
   145                2 :         vma->vm_ops = &ext4_file_vm_ops;
   146                2 :         vma->vm_flags |= VM_CAN_NONLINEAR;
   147                2 :         return 0;
   148                  : }
   149                  :
   150                  : static int ext4_file_open(struct inode * inode, struct file * filp)
   151         48757716 : {
   152         48757716 :         struct super_block *sb = inode->i_sb;
   153         97515432 :         struct ext4_sb_info *sbi = EXT4_SB(inode->i_sb);
   154         48757716 :         struct vfsmount *mnt = filp->f_path.mnt;
   155                  :         struct path path;
   156                  :         char buf[64], *cp;
   157                  :
   158         48757716 :         if (unlikely(!(sbi->s_mount_flags & EXT4_MF_MNTDIR_SAMPLED) &&
   159                  :                      !(sb->s_flags & MS_RDONLY))) {
   160               74 :                 sbi->s_mount_flags |= EXT4_MF_MNTDIR_SAMPLED;
   161                  :                 /*
   162                  :                  * Sample where the filesystem has been mounted and
   163                  :                  * store it in the superblock for sysadmin convenience
   164                  :                  * when trying to sort through large numbers of block
   165                  :                  * devices or filesystem images.
   166                  :                  */
   167                  :                 memset(buf, 0, sizeof(buf));
   168               74 :                 path.mnt = mnt->mnt_parent;
   169               74 :                 path.dentry = mnt->mnt_mountpoint;
   170               74 :                 path_get(&path);
   171               74 :                 cp = d_path(&path, buf, sizeof(buf));
   172               74 :                 path_put(&path);
   173               74 :                 if (!IS_ERR(cp)) {
   174               74 :                         memcpy(sbi->s_es->s_last_mounted, cp,
   175                  :                                sizeof(sbi->s_es->s_last_mounted));
   176               74 :                         sb->s_dirt = 1;
   177                  :                 }
   178                  :         }
   179         48745975 :         return generic_file_open(inode, filp);
   180                  : }
   181                  :
   182                  : const struct file_operations ext4_file_operations = {
   183                  :         .llseek         = generic_file_llseek,
   184                  :         .read           = do_sync_read,
   185                  :         .write          = do_sync_write,
   186                  :         .aio_read       = generic_file_aio_read,
   187                  :         .aio_write      = ext4_file_write,
   188                  :         .unlocked_ioctl = ext4_ioctl,
   189                  : #ifdef CONFIG_COMPAT
   190                  :         .compat_ioctl   = ext4_compat_ioctl,
   191                  : #endif
   192                  :         .mmap           = ext4_file_mmap,
   193                  :         .open           = ext4_file_open,
   194                  :         .release        = ext4_release_file,
   195                  :         .fsync          = ext4_sync_file,
   196                  :         .splice_read    = generic_file_splice_read,
   197                  :         .splice_write   = generic_file_splice_write,
   198                  : };
   199                  :
```

```
200                 : const struct inode_operations ext4_file_inode_operations = {
201                 :         .truncate       = ext4_truncate,
202                 :         .setattr        = ext4_setattr,
203                 :         .getattr        = ext4_getattr,
204                 : #ifdef CONFIG_EXT4_FS_XATTR
205                 :         .setxattr       = generic_setxattr,
206                 :         .getxattr       = generic_getxattr,
207                 :         .listxattr      = ext4_listxattr,
208                 :         .removexattr    = generic_removexattr,
209                 : #endif
210                 :         .permission     = ext4_permission,
211                 :         .fallocate      = ext4_fallocate,
212                 :         .fiemap         = ext4_fiemap,
213                 : };
214                 :
```