LCOV - code coverage report

Current view:	directory - fs/ext4 - acl.c (source / functions)		Found	Hit	Coverage
Test:	kernel_2_6_31_ext4_round_3.info	Lines:	248	17	6.9 %
Date:	2009-10-24	Functions:	16	3	18.8 %

```
      1                :  /*
      2                :   * linux/fs/ext4/acl.c
      3                :   *
      4                :   * Copyright (C) 2001-2003 Andreas Gruenbacher, <agruen@suse.de>
      5                :   */
      6                :
      7                :  #include <linux/init.h>
      8                :  #include <linux/sched.h>
      9                :  #include <linux/slab.h>
     10                :  #include <linux/capability.h>
     11                :  #include <linux/fs.h>
     12                :  #include "ext4_jbd2.h"
     13                :  #include "ext4.h"
     14                :  #include "xattr.h"
     15                :  #include "acl.h"
     16                :
     17                :  /*
     18                :   * Convert from filesystem to in-memory representation.
     19                :   */
     20                :  static struct posix_acl *
     21                :  ext4_acl_from_disk(const void *value, size_t size)
     22              0 :  {
     23              0 :          const char *end = (char *)value + size;
     24                :          int n, count;
     25                :          struct posix_acl *acl;
     26                :
     27              0 :          if (!value)
     28              0 :                  return NULL;
     29              0 :          if (size < sizeof(ext4_acl_header))
     30              0 :                   return ERR_PTR(-EINVAL);
     31              0 :          if (((ext4_acl_header *)value)->a_version !=
     32                :              cpu_to_le32(EXT4_ACL_VERSION))
     33              0 :                  return ERR_PTR(-EINVAL);
     34              0 :          value = (char *)value + sizeof(ext4_acl_header);
     35              0 :          count = ext4_acl_count(size);
     36              0 :          if (count < 0)
     37              0 :                  return ERR_PTR(-EINVAL);
     38              0 :          if (count == 0)
     39              0 :                  return NULL;
     40              0 :          acl = posix_acl_alloc(count, GFP_NOFS);
     41              0 :          if (!acl)
     42              0 :                  return ERR_PTR(-ENOMEM);
     43              0 :          for (n = 0; n < count; n++) {
     44                :                  ext4_acl_entry *entry =
     45              0 :                          (ext4_acl_entry *)value;
     46              0 :                  if ((char *)value + sizeof(ext4_acl_entry_short) > end)
     47              0 :                          goto fail;
     48              0 :                  acl->a_entries[n].e_tag  = le16_to_cpu(entry->e_tag);
     49              0 :                  acl->a_entries[n].e_perm = le16_to_cpu(entry->e_perm);
     50                :
     51              0 :                  switch (acl->a_entries[n].e_tag) {
     52                :                  case ACL_USER_OBJ:
     53                :                  case ACL_GROUP_OBJ:
     54                :                  case ACL_MASK:
     55                :                  case ACL_OTHER:
     56              0 :                          value = (char *)value +
     57                :                                  sizeof(ext4_acl_entry_short);
```

```
    58                0 :                             acl->a_entries[n].e_id = ACL_UNDEFINED_ID;
    59                0 :                             break;
    60                  :
    61                  :                     case ACL_USER:
    62                  :                     case ACL_GROUP:
    63                0 :                             value = (char *)value + sizeof(ext4_acl_entry);
    64                0 :                             if ((char *)value > end)
    65                0 :                                     goto fail;
    66                0 :                             acl->a_entries[n].e_id =
    67                  :                                     le32_to_cpu(entry->e_id);
    68                0 :                             break;
    69                  :
    70                  :                     default:
    71                  :                             goto fail;
    72                  :                     }
    73                  :             }
    74                0 :             if (value != end)
    75                0 :                     goto fail;
    76                0 :             return acl;
    77                  :
    78                0 : fail:
    79                0 :             posix_acl_release(acl);
    80                0 :             return ERR_PTR(-EINVAL);
    81                  : }
    82                  :
    83                  : /*
    84                  :  * Convert from in-memory to filesystem representation.
    85                  :  */
    86                  : static void *
    87                  : ext4_acl_to_disk(const struct posix_acl *acl, size_t *size)
    88                0 : {
    89                  :             ext4_acl_header *ext_acl;
    90                  :             char *e;
    91                  :             size_t n;
    92                  :
    93                0 :             *size = ext4_acl_size(acl->a_count);
    94                0 :             ext_acl = kmalloc(sizeof(ext4_acl_header) + acl->a_count *
    95                  :                             sizeof(ext4_acl_entry), GFP_NOFS);
    96                0 :             if (!ext_acl)
    97                0 :                     return ERR_PTR(-ENOMEM);
    98                0 :             ext_acl->a_version = cpu_to_le32(EXT4_ACL_VERSION);
    99                0 :             e = (char *)ext_acl + sizeof(ext4_acl_header);
   100                0 :             for (n = 0; n < acl->a_count; n++) {
   101                0 :                     ext4_acl_entry *entry = (ext4_acl_entry *)e;
   102                0 :                     entry->e_tag  = cpu_to_le16(acl->a_entries[n].e_tag);
   103                0 :                     entry->e_perm = cpu_to_le16(acl->a_entries[n].e_perm);
   104                0 :                     switch (acl->a_entries[n].e_tag) {
   105                  :                     case ACL_USER:
   106                  :                     case ACL_GROUP:
   107                0 :                             entry->e_id = cpu_to_le32(acl->a_entries[n].e_id);
   108                0 :                             e += sizeof(ext4_acl_entry);
   109                0 :                             break;
   110                  :
   111                  :                     case ACL_USER_OBJ:
   112                  :                     case ACL_GROUP_OBJ:
   113                  :                     case ACL_MASK:
   114                  :                     case ACL_OTHER:
   115                0 :                             e += sizeof(ext4_acl_entry_short);
   116                0 :                             break;
   117                  :
   118                  :                     default:
   119                  :                             goto fail;
   120                  :                     }
   121                  :             }
   122                0 :             return (char *)ext_acl;
   123                  :
   124                0 : fail:
   125                0 :             kfree(ext_acl);
   126                0 :             return ERR_PTR(-EINVAL);
   127                  : }
   128                  :
```

```
   129                :  /*
   130                :   * Inode operation get_posix_acl().
   131                :   *
   132                :   * inode->i_mutex: don't care
   133                :   */
   134                :  static struct posix_acl *
   135                :  ext4_get_acl(struct inode *inode, int type)
   136              0 :  {
   137                :          int name_index;
   138              0 :          char *value = NULL;
   139                :          struct posix_acl *acl;
   140                :          int retval;
   141                :
   142              0 :          if (!test_opt(inode->i_sb, POSIX_ACL))
   143              0 :                  return NULL;
   144                :
   145              0 :          acl = get_cached_acl(inode, type);
   146              0 :          if (acl != ACL_NOT_CACHED)
   147              0 :                  return acl;
   148                :
   149              0 :          switch (type) {
   150                :          case ACL_TYPE_ACCESS:
   151              0 :                  name_index = EXT4_XATTR_INDEX_POSIX_ACL_ACCESS;
   152              0 :                  break;
   153                :          case ACL_TYPE_DEFAULT:
   154              0 :                  name_index = EXT4_XATTR_INDEX_POSIX_ACL_DEFAULT;
   155              0 :                  break;
   156                :          default:
   157              0 :                  BUG();
   158                :          }
   159              0 :          retval = ext4_xattr_get(inode, name_index, "", NULL, 0);
   160              0 :          if (retval > 0) {
   161              0 :                  value = kmalloc(retval, GFP_NOFS);
   162              0 :                  if (!value)
   163              0 :                          return ERR_PTR(-ENOMEM);
   164              0 :                  retval = ext4_xattr_get(inode, name_index, "", value, retval);
   165                :          }
   166              0 :          if (retval > 0)
   167              0 :                  acl = ext4_acl_from_disk(value, retval);
   168              0 :          else if (retval == -ENODATA || retval == -ENOSYS)
   169              0 :                  acl = NULL;
   170                :          else
   171              0 :                  acl = ERR_PTR(retval);
   172              0 :          kfree(value);
   173                :
   174              0 :          if (!IS_ERR(acl))
   175              0 :                  set_cached_acl(inode, type, acl);
   176                :
   177              0 :          return acl;
   178                :  }
   179                :
   180                :  /*
   181                :   * Set the access or default ACL of an inode.
   182                :   *
   183                :   * inode->i_mutex: down unless called from ext4_new_inode
   184                :   */
   185                :  static int
   186                :  ext4_set_acl(handle_t *handle, struct inode *inode, int type,
   187                :               struct posix_acl *acl)
   188              0 :  {
   189                :          int name_index;
   190              0 :          void *value = NULL;
   191              0 :          size_t size = 0;
   192                :          int error;
   193                :
   194              0 :          if (S_ISLNK(inode->i_mode))
   195              0 :                  return -EOPNOTSUPP;
   196                :
   197              0 :          switch (type) {
   198                :          case ACL_TYPE_ACCESS:
   199              0 :                  name_index = EXT4_XATTR_INDEX_POSIX_ACL_ACCESS;
```

```
   200                0 :                     if (acl) {
   201                0 :                             mode_t mode = inode->i_mode;
   202                0 :                             error = posix_acl_equiv_mode(acl, &mode);
   203                0 :                             if (error < 0)
   204                0 :                                     return error;
   205                  :                             else {
   206                0 :                                     inode->i_mode = mode;
   207                0 :                                     ext4_mark_inode_dirty(handle, inode);
   208                0 :                                     if (error == 0)
   209                0 :                                             acl = NULL;
   210                  :                             }
   211                  :                     }
   212                  :                     break;
   213                  :
   214                  :             case ACL_TYPE_DEFAULT:
   215                0 :                     name_index = EXT4_XATTR_INDEX_POSIX_ACL_DEFAULT;
   216                0 :                     if (!S_ISDIR(inode->i_mode))
   217                0 :                             return acl ? -EACCES : 0;
   218                  :                     break;
   219                  :
   220                  :             default:
   221                0 :                     return -EINVAL;
   222                  :             }
   223                0 :             if (acl) {
   224                0 :                     value = ext4_acl_to_disk(acl, &size);
   225                0 :                     if (IS_ERR(value))
   226                0 :                             return (int)PTR_ERR(value);
   227                  :             }
   228                  :
   229                0 :             error = ext4_xattr_set_handle(handle, inode, name_index, "",
   230                  :                                           value, size, 0);
   231                  :
   232                0 :             kfree(value);
   233                0 :             if (!error)
   234                0 :                     set_cached_acl(inode, type, acl);
   235                  :
   236                0 :             return error;
   237                  : }
   238                  :
   239                  : static int
   240                  : ext4_check_acl(struct inode *inode, int mask)
   241                0 : {
   242                0 :             struct posix_acl *acl = ext4_get_acl(inode, ACL_TYPE_ACCESS);
   243                  :
   244                0 :             if (IS_ERR(acl))
   245                0 :                     return PTR_ERR(acl);
   246                0 :             if (acl) {
   247                0 :                     int error = posix_acl_permission(inode, acl, mask);
   248                0 :                     posix_acl_release(acl);
   249                0 :                     return error;
   250                  :             }
   251                  :
   252                0 :             return -EAGAIN;
   253                  : }
   254                  :
   255                  : int
   256                  : ext4_permission(struct inode *inode, int mask)
   257        300474422 : {
   258        300474422 :             return generic_permission(inode, mask, ext4_check_acl);
   259                  : }
   260                  :
   261                  : /*
   262                  :  * Initialize the ACLs of a new inode. Called from ext4_new_inode.
   263                  :  *
   264                  :  * dir->i_mutex: down
   265                  :  * inode->i_mutex: up (access to inode is still exclusive)
   266                  :  */
   267                  : int
   268                  : ext4_init_acl(handle_t *handle, struct inode *inode, struct inode *dir)
   269          9312572 : {
   270          9312572 :             struct posix_acl *acl = NULL;
```

```
   271  9312572 :         int error = 0;
   272          :
   273  9312572 :         if (!S_ISLNK(inode->i_mode)) {
   274 18625144 :                 if (test_opt(dir->i_sb, POSIX_ACL)) {
   275        0 :                         acl = ext4_get_acl(dir, ACL_TYPE_DEFAULT);
   276        0 :                         if (IS_ERR(acl))
   277        0 :                                 return PTR_ERR(acl);
   278          :                 }
   279  9312572 :                 if (!acl)
   280  9312572 :                         inode->i_mode &= ~current_umask();
   281          :         }
   282 18625144 :         if (test_opt(inode->i_sb, POSIX_ACL) && acl) {
   283          :                 struct posix_acl *clone;
   284          :                 mode_t mode;
   285          :
   286        0 :                 if (S_ISDIR(inode->i_mode)) {
   287        0 :                         error = ext4_set_acl(handle, inode,
   288          :                                              ACL_TYPE_DEFAULT, acl);
   289        0 :                         if (error)
   290        0 :                                 goto cleanup;
   291          :                 }
   292        0 :                 clone = posix_acl_clone(acl, GFP_NOFS);
   293        0 :                 error = -ENOMEM;
   294        0 :                 if (!clone)
   295        0 :                         goto cleanup;
   296          :
   297        0 :                 mode = inode->i_mode;
   298        0 :                 error = posix_acl_create_masq(clone, &mode);
   299        0 :                 if (error >= 0) {
   300        0 :                         inode->i_mode = mode;
   301        0 :                         if (error > 0) {
   302          :                                 /* This is an extended ACL */
   303        0 :                                 error = ext4_set_acl(handle, inode,
   304          :                                                      ACL_TYPE_ACCESS, clone);
   305          :                         }
   306          :                 }
   307        0 :                 posix_acl_release(clone);
   308          :         }
   309  9312572 : cleanup:
   310  9312572 :         posix_acl_release(acl);
   311  9312572 :         return error;
   312          : }
   313          :
   314          : /*
   315          :  * Does chmod for an inode that may have an Access Control List. The
   316          :  * inode->i_mode field must be updated to the desired value by the caller
   317          :  * before calling this function.
   318          :  * Returns 0 on success, or a negative error number.
   319          :  *
   320          :  * We change the ACL rather than storing some ACL entries in the file
   321          :  * mode permission bits (which would be more efficient), because that
   322          :  * would break once additional permissions (like  ACL_APPEND, ACL_DELETE
   323          :  * for directories) are added. There are no more bits available in the
   324          :  * file mode.
   325          :  *
   326          :  * inode->i_mutex: down
   327          :  */
   328          : int
   329          : ext4_acl_chmod(struct inode *inode)
   330        1 : {
   331          :         struct posix_acl *acl, *clone;
   332          :         int error;
   333          :
   334        1 :         if (S_ISLNK(inode->i_mode))
   335        0 :                 return -EOPNOTSUPP;
   336        2 :         if (!test_opt(inode->i_sb, POSIX_ACL))
   337        1 :                 return 0;
   338        0 :         acl = ext4_get_acl(inode, ACL_TYPE_ACCESS);
   339        0 :         if (IS_ERR(acl) || !acl)
   340        0 :                 return PTR_ERR(acl);
   341        0 :         clone = posix_acl_clone(acl, GFP_KERNEL);
```

```
342                0 :         posix_acl_release(acl);
343                0 :         if (!clone)
344                0 :                 return -ENOMEM;
345                0 :         error = posix_acl_chmod_masq(clone, inode->i_mode);
346                0 :         if (!error) {
347                  :                 handle_t *handle;
348                0 :                 int retries = 0;
349                  :
350                0 :         retry:
351                0 :                 handle = ext4_journal_start(inode,
352                  :                                             EXT4_DATA_TRANS_BLOCKS(inode->i_sb));
353                0 :                 if (IS_ERR(handle)) {
354                0 :                         error = PTR_ERR(handle);
355                0 :                         ext4_std_error(inode->i_sb, error);
356                  :                         goto out;
357                  :                 }
358                0 :                 error = ext4_set_acl(handle, inode, ACL_TYPE_ACCESS, clone);
359                0 :                 ext4_journal_stop(handle);
360                0 :                 if (error == -ENOSPC &&
361                  :                     ext4_should_retry_alloc(inode->i_sb, &retries))
362                0 :                         goto retry;
363                  :         }
364                0 : out:
365                0 :         posix_acl_release(clone);
366                0 :         return error;
367                  : }
368                  :
369                  : /*
370                  :  * Extended attribute handlers
371                  :  */
372                  : static size_t
373                  : ext4_xattr_list_acl_access(struct inode *inode, char *list, size_t list_len,
374                  :                            const char *name, size_t name_len)
375                0 : {
376                0 :         const size_t size = sizeof(POSIX_ACL_XATTR_ACCESS);
377                  :
378                0 :         if (!test_opt(inode->i_sb, POSIX_ACL))
379                0 :                 return 0;
380                0 :         if (list && size <= list_len)
381                  :                 memcpy(list, POSIX_ACL_XATTR_ACCESS, size);
382                0 :         return size;
383                  : }
384                  :
385                  : static size_t
386                  : ext4_xattr_list_acl_default(struct inode *inode, char *list, size_t list_len,
387                  :                             const char *name, size_t name_len)
388                0 : {
389                0 :         const size_t size = sizeof(POSIX_ACL_XATTR_DEFAULT);
390                  :
391                0 :         if (!test_opt(inode->i_sb, POSIX_ACL))
392                0 :                 return 0;
393                0 :         if (list && size <= list_len)
394                  :                 memcpy(list, POSIX_ACL_XATTR_DEFAULT, size);
395                0 :         return size;
396                  : }
397                  :
398                  : static int
399                  : ext4_xattr_get_acl(struct inode *inode, int type, void *buffer, size_t size)
400                0 : {
401                  :         struct posix_acl *acl;
402                  :         int error;
403                  :
404                0 :         if (!test_opt(inode->i_sb, POSIX_ACL))
405                0 :                 return -EOPNOTSUPP;
406                  :
407                0 :         acl = ext4_get_acl(inode, type);
408                0 :         if (IS_ERR(acl))
409                0 :                 return PTR_ERR(acl);
410                0 :         if (acl == NULL)
411                0 :                 return -ENODATA;
412                0 :         error = posix_acl_to_xattr(acl, buffer, size);
```

```
   413                0 :         posix_acl_release(acl);
   414                  :
   415                0 :         return error;
   416                  : }
   417                  :
   418                  : static int
   419                  : ext4_xattr_get_acl_access(struct inode *inode, const char *name,
   420                  :                           void *buffer, size_t size)
   421                0 : {
   422                0 :         if (strcmp(name, "") != 0)
   423                0 :                 return -EINVAL;
   424                0 :         return ext4_xattr_get_acl(inode, ACL_TYPE_ACCESS, buffer, size);
   425                  : }
   426                  :
   427                  : static int
   428                  : ext4_xattr_get_acl_default(struct inode *inode, const char *name,
   429                  :                            void *buffer, size_t size)
   430                0 : {
   431                0 :         if (strcmp(name, "") != 0)
   432                0 :                 return -EINVAL;
   433                0 :         return ext4_xattr_get_acl(inode, ACL_TYPE_DEFAULT, buffer, size);
   434                  : }
   435                  :
   436                  : static int
   437                  : ext4_xattr_set_acl(struct inode *inode, int type, const void *value,
   438                  :                    size_t size)
   439                0 : {
   440                  :         handle_t *handle;
   441                  :         struct posix_acl *acl;
   442                0 :         int error, retries = 0;
   443                  :
   444                0 :         if (!test_opt(inode->i_sb, POSIX_ACL))
   445                0 :                 return -EOPNOTSUPP;
   446                0 :         if (!is_owner_or_cap(inode))
   447                0 :                 return -EPERM;
   448                  :
   449                0 :         if (value) {
   450                0 :                 acl = posix_acl_from_xattr(value, size);
   451                0 :                 if (IS_ERR(acl))
   452                0 :                         return PTR_ERR(acl);
   453                0 :                 else if (acl) {
   454                0 :                         error = posix_acl_valid(acl);
   455                0 :                         if (error)
   456                0 :                                 goto release_and_out;
   457                  :                 }
   458                  :         } else
   459                0 :                 acl = NULL;
   460                  :
   461                0 : retry:
   462                0 :         handle = ext4_journal_start(inode, EXT4_DATA_TRANS_BLOCKS(inode->i_sb));
   463                0 :         if (IS_ERR(handle))
   464                0 :                 return PTR_ERR(handle);
   465                0 :         error = ext4_set_acl(handle, inode, type, acl);
   466                0 :         ext4_journal_stop(handle);
   467                0 :         if (error == -ENOSPC && ext4_should_retry_alloc(inode->i_sb, &retries))
   468                0 :                 goto retry;
   469                  :
   470                0 : release_and_out:
   471                0 :         posix_acl_release(acl);
   472                0 :         return error;
   473                  : }
   474                  :
   475                  : static int
   476                  : ext4_xattr_set_acl_access(struct inode *inode, const char *name,
   477                  :                           const void *value, size_t size, int flags)
   478                0 : {
   479                0 :         if (strcmp(name, "") != 0)
   480                0 :                 return -EINVAL;
   481                0 :         return ext4_xattr_set_acl(inode, ACL_TYPE_ACCESS, value, size);
   482                  : }
   483                  :
```

```
   484                 : static int
   485                 : ext4_xattr_set_acl_default(struct inode *inode, const char *name,
   486                 :                            const void *value, size_t size, int flags)
   487               0 : {
   488               0 :         if (strcmp(name, "") != 0)
   489               0 :                 return -EINVAL;
   490               0 :         return ext4_xattr_set_acl(inode, ACL_TYPE_DEFAULT, value, size);
   491                 : }
   492                 :
   493                 : struct xattr_handler ext4_xattr_acl_access_handler = {
   494                 :         .prefix = POSIX_ACL_XATTR_ACCESS,
   495                 :         .list   = ext4_xattr_list_acl_access,
   496                 :         .get    = ext4_xattr_get_acl_access,
   497                 :         .set    = ext4_xattr_set_acl_access,
   498                 : };
   499                 :
   500                 : struct xattr_handler ext4_xattr_acl_default_handler = {
   501                 :         .prefix = POSIX_ACL_XATTR_DEFAULT,
   502                 :         .list   = ext4_xattr_list_acl_default,
   503                 :         .get    = ext4_xattr_get_acl_default,
   504                 :         .set    = ext4_xattr_set_acl_default,
   505                 : };
```