Detailed Level Design of v2 for Remote ACL

Fan Yong

2007-09-03

1 Functional Specification

1.1 User level utils for remote_acl

The "lfs {l,r}{s,g}etfacl" utils ioctl lustre mountpoint with the following flags to notify LLITE kernel which acl operation will be done.

```
enum {
        RMT_LSETFACL    = 1,
        RMT_LGETFACL    = 2,
        RMT_RSETFACL    = 3,
        RMT_RGETFACL    = 4
};
```

1.1.1 System getfacl utils output filter for "lfs rgetfacl"

For "lfs rgetfacl", the system getfacl utils execed by "lfs rgetfacl" child process converts server-side uid/gid in the result ACL items to username/groupname based on client-side user database before outputting to the user. But it maybe misguide the user. So such output will be converted back to uid/gid based on the same client-side user database by "lfs rgetfacl" parent process before real output.

- Prototype:

  ```
  int rgetfacl_output(char *str)
  ```

- Parameter:

  ```
  str: the string which will be converted.
  ```

- Return value:

  ```
  >= 0: the converted output string length.
  < 0: conversion failed.
  ```

- Description:

  ```
  This function scans the input string, and converts username/groupname to uid/gid based
  ```

1.1.2 The "lfs {l,r}{s,g}etfacl" implementation function – do_rmtacl

- Prototype:

```
int do_rmtacl(int argc, char *argv[], int ops, int (output_func)(char *))
```

- Parameter:

```
argc: parameter count for system {s,g}etfacl utils.
argv: parameter array for system {s,g}etfacl utils.
ops: for notifying LLITE kernel which acl operation will be done.
output_func: the filter function for output.
```

- Return value:

```
-1: failed.
others: the same as system {s,g}etfacl utils return value.
```

- Description:

```
This function ioctls lustre mountpoint to notify LLITE kernel which acl operation will
```

1.1.3 "lfs lsetfacl" interface.

- Prototype:

```
int llapi_lsetfacl(int argc, char *argv[])
```

- Parameter:

```
argc: parameter count for system setfacl utils.
argv: parameter array for system setfacl utils, argv[0] is "setfacl".
```

- Return value:

```
The same as "do_rmtacl" return value.
```

- Description:

```
This function calls "do_rmtacl(argc, argv, RMTLSETFACL, NULL);".
```

1.1.4 "lfs lgetfacl" interface.

- Prototype:

```
int llapi_lgetfacl(int argc, char *argv[])
```

- Parameter:

  argc: parameter count for system getfacl utils.
  argv: parameter array for system getfacl utils, argv[0] is "getfacl".

- Return value:

  The same as "do_rmtacl" return value.

- Description:

  This function calls "do_rmtacl(argc, argv, RMTLGETFACL, NULL);".

1.1.5 "lfs rsetfacl" interface.

- Prototype:

  ```
  int llapi_rsetfacl(int argc, char *argv[])
  ```

- Parameter:

  argc: parameter count for system setfacl utils.
  argv: parameter array for system setfacl utils, argv[0] is "setfacl".

- Return value:

  The same as "do_rmtacl" return value.

- Description:

  This function calls "do_rmtacl(argc, argv, RMTRSETFACL, NULL);".

1.1.6 "lfs rgetfacl" interface.

- Prototype:

  ```
  int llapi_rgetfacl(int argc, char *argv[])
  ```

- Parameter:

  argc: parameter count for system getfacl utils.
  argv: parameter array for system getfacl utils, argv[0] is "getfacl".

- Return value:

  The same as "do_rmtacl" return value.

- Description:

  This function calls "do_rmtacl(argc, argv, RMTRGETFACL, rgetfacl_output);".

1.2 Idmap related process in OBDCLASS

1.2.1 Define idmap related struct

```
enum {
    CFS_IDMAP_HASHSIZE      = 32
};
enum lustre_idmap_idx {
    RMT_UIDMAP_IDX,
    LCL_UIDMAP_IDX,
    RMT_GIDMAP_IDX,
    LCL_GIDMAP_IDX,
    CFS_IDMAP_N_HASHES
};
struct lustre_idmap_entry {
    struct list_head lie_rmt_uid_hash; /* hashed as lie_rmt_uid; */
    struct list_head lie_lcl_uid_hash; /* hashed as lie_lcl_uid; */
    struct list_head lie_rmt_gid_hash; /* hashed as lie_rmt_gid; */
    struct list_head lie_lcl_gid_hash; /* hashed as lie_lcl_gid; */
    uid_t            lie_rmt_uid;      /* remote uid */
    uid_t            lie_lcl_uid;      /* local uid */
    gid_t            lie_rmt_gid;      /* remote gid */
    gid_t            lie_lcl_gid;      /* local gid */
};
struct lustre_idmap_table {
    spinlock_t lit_lock;
    struct list_head lit_idmaps[CFS_IDMAP_N_HASHES][CFS_IDMAP_HASHSIZE];
};
```

This is the mapping rules:

- It is a quarternary combinatory for idmap entry.
- The quarternary combinatory is unique in the idmap table.
- The quarternary combinatory consists of two parts: uid pair and gid pair.
- For each pair, it is N(client-side) : 1(server-side) mapping.
- For MDS_CONNECT / SEC_CTX_INIT, insert idmap entry into idmap table, if the idmap entry already exists, do nothing.
- For MDS_DISCONNECT / SEC_CTX_FINI, remove idmap entry from idmap table, if the idmap entry doesn't exist, do nothing.
- No need "lie_refcount" anymore, for resending MDS_CONNECT / SEC_CTX_INIT / MDS_DISCONNECT / SEC_CTX_FINI maybe cause "lie_refcount" in confusion and meaningless.

- No use binary combinatory of idmap entry anymore, for uid and gid are N : N related, new uid pair doesn't mean new gid pair, the same for the contrary case. It is difficult to process insert / remove idmap entry without "lie_refcount" in binary combinatory of idmap entry.

1.2.2 lustre_idmap_init

- Prototype:

  ```
  struct lustre_idmap_table *lustre_idmap_init(void)
  ```

- Parameter:

  ```
  Nothing.
  ```

- Return value:

  ```
  -ENOMEM: not enough memory.
  valid pointer: pointer to the idmap table generated.
  ```

- Description:

  ```
  This function initializes the idmap table. It is not a new one, just moved from MDT to
  ```

1.2.3 lustre_idmap_fini

- Prototype:

  ```
  void lustre_idmap_fini(struct lustre_idmap_table *t)
  ```

- Parameter:

  ```
  t: pointer to the idmap table.
  ```

- Return value:

  ```
  Nothing.
  ```

- Description:

  ```
  This function cleanups the idmap table, including releases all the idmap items. It is n
  ```

1.2.4 lustre_idmap_add

- Prototype:

```
int lustre_idmap_add(struct lustre_idmap_table *t, uid_t ruid, uid_t luid, gid_t rgid,
                     gid_t lgid)
```

- Parameter:

```
t: pointer to the user idmap table.
ruid: client-side uid.
luid: server-side uid.
rgid: client-side gid.
lgid: server-side gid.
```

- Return value:

```
0: succeed.
-ENOMEM: not enough memory.
-EACCESS: invalid mapping item.
others: other fail.
```

- Description:

```
This function inserts the idmap item "ruid/rgid <=> luid/lgid" into the idmap table if
```

1.2.5 lustre_idmap_del

- Prototype:

```
int lustre_idmap_del(struct lustre_idmap_table *t, uid_t ruid, uid_t luid, gid_t rgid,
                     gid_t lgid)
```

- Parameter:

```
t: pointer to the user idmap table.
ruid: client-side uid.
luid: server-side uid.
rgid: client-side gid.
lgid: server-side gid.
```

- Return value:

```
0: succeed.
-EACCESS: invalid mapping item.
others: other fail.
```

- Description:

```
This function removes the idmap item "ruid/rgid <=> luid/lgid" from the idmap table. It
```

1.2.6 lustre_idmap_lookup_uid

- Prototype:

```
int lustre_idmap_lookup_uid(struct md_ucred *mu, struct lustre_idmap_table *t, int reve
                            uid_t uid)
```

- Parameter:

```
mu: pointer to the user credential struct.
t: pointer to the user idmap table.
reverse: flags for client-side to server-side mapping or the contrary case.
uid: the original uid for lookup.
```

- Return value:

```
CFS_IDMAP_NOTFOUND: uid is not mapped.
others: the mapped uid.
```

- Description:

```
This function searches the "mu" or the uid mapping table with the specified "uid", and
```

1.2.7 lustre_idmap_lookup_gid

- Prototype:

```
int lustre_idmap_lookup_gid(struct md_ucred *mu, struct lustre_idmap_table *t, int reve
                            gid_t gid)
```

- Parameter:

```
mu: pointer to the user credential struct.
t: pointer to the user idmap table.
reverse: flags for client-side to server-side mapping or the contrary case.
gid: the original gid for lookup.
```

- Return value:

```
CFS_IDMAP_NOTFOUND: gid is not mapped.
others: the mapped gid.
```

- Description:

```
This function searches the "mu" or the gid mapping table with the specified "gid", and
```

1.3 ACL related process in OBDCLASS

1.3.1 Define extended acl_xattr struct

Extend "posix_acl_xattr_entry" with new member "e_stat" which records the stat of such ACL entry:

```
enum {
        ES_UNK  = 0,    /* unknown stat */
        ES_UNC  = 1,    /* ACL entry is not changed */
        ES_MOD  = 2,    /* ACL entry is modified */
        ES_ADD  = 3,    /* ACL entry is added */
        ES_DEL  = 4     /* ACL entry is deleted */
};
typedef struct {
        __u16 e_tag;    /* the same as posix_acl_xattr_entry.e_tag */
        __u16 e_perm;   /* the same as posix_acl_xattr_entry.e_perm */
        __u32 e_id;     /* the same as posix_acl_xattr_entry.e_id */
        __u32 e_stat;   /* status of eacl entry: ES_UNK/ES_UNC/ES_MOD/ES_ADD/ES_DEL */
} ext_acl_xattr_entry;
typedef struct {
        __u32               a_count;
        ext_acl_xattr_entry a_entries[0];
} ext_acl_xattr_header;
```

1.3.2 lustre_posix_acl_permission

- Prototype:

```
int lustre_posix_acl_permission(struct md_ucred *mu, struct lu_attr *la, int want,
                                posix_acl_xattr_entry *entry, int count)
```

- Parameter:

```
mu: pointer to the user credential struct.
la: pointer to the object's attribute.
want: permission mask to be checked.
entry: pointer to the posix ACL.
count: count of posix ACL items
```

* Return value:

```
0: pass permission check.
-EACCES: permission denied.
-EIO: invalid posix ACL.
```

- Description:

```
This function checks permission based on posix ACL. It is not a new one, just moved fro
```

1.3.3 lustre_posix_acl_chmod_masq

- Prototype:

```
int lustre_posix_acl_chmod_masq(posix_acl_xattr_entry *entry, __u32 mode, int count)
```

- Parameter:

```
entry: pointer to the posix ACL.
mode: the mode parameter for chmod operation.
count: count of the posix ACL items.
```

- Return value:

```
0: success
-EIO: invalid posix ACL.
```

- Description:

```
This function fixes the posix ACL based on the "mode" for chmod operation. It is not a
```

1.3.4 lustre_posix_acl_create_masq

- Prototype:

```
int lustre_posix_acl_create_masq(posix_acl_xattr_entry *entry, __u32 *pmode, int count)
```

- Parameter:

```
entry: pointer to the posix ACL.
pmode: pointer to the mode parameter for open/creat operation.
count: count of posix ACL items.
```

- Return value:

```
1: input posix ACL is fixed.
0: input posix ACL is not fixed.
-EIO: invalid posix ACL.
```

- Description:

```
This function scans the posix ACL, all permissions in *pmode that are not granted by th
```

1.3.5 lustre_posix_acl_xattr_2ext

- Prototype:

```
ext_acl_xattr_header *lustre_posix_acl_xattr_2ext(posix_acl_xattr_header *header,
                                                  int size)
```

- Parameter:

```
header: pointer to the posix ACL head.
size: size of posix ACL.
```

- Return value:

```
valid pointer: pointer to the extended ACL generated.
-EINVAL: invalid posix ACL size.
-ENOMEM: not enough memory.
```

- Description:

```
This function generates new extended ACL based on the input posix ACL.
```

1.3.6 lustre_posix_acl_xattr_filter

- Prototype:

```
int lustre_posix_acl_xattr_filter(posix_acl_xattr_header *header, int size,
                                  posix_acl_xattr_header **out)
```

- Parameter:

```
header: pointer to the posix ACL head.
size: size of posix ACL.
out: double pointer to the posix ACL generated.
```

- Return value:

```
>= 0: size of the generated posix ACL.
-EINVAL: invalid posix ACL size.
-ENOMEM: not enough memory.
-EIO: invalid posix ACL.
```

- Description:

```
This function generates new posix ACL based on all "non-nobody" items.
```

1.3.7 lustre_posix_acl_xattr_id2client

- Prototype:

```
int lustre_posix_acl_xattr_id2client(struct md_ucred *mu, struct lustre_idmap_table *t,
                                     posix_acl_xattr_header *header, int size, int flags
```

- Parameter:

```
mu: pointer to the user credential struct.
t: pointer to the user idmap table.
header: pointer to the posix ACL head.
size: count of posix ACL.
flags: which posix ACL items need to be converted.
      enum {
          CFS_IC_NOTHING   = 0,    /* convert nothing */
          CFS_IC_ALL       = 1,    /* convert all items */
          CFS_IC_MAPPED    = 2,    /* convert items containning mapped uid/gid */
          CFS_IC_UNMAPPED  = 3    /* convert items containning unmapped uid/gid */
      };
```

- Return value:

```
0: convert succeed.
-EINVAL: invalid posix ACL size.
-EIO: invalid posix ACL.
```

- Description:

```
This function converts server-side uid/gid in the posix ACL items to the client-side on
```

1.3.8 lustre_posix_acl_xattr_free

- Prototype:

```
void lustre_posix_acl_xattr_free(posix_acl_xattr_header *header, int size)
```

- Parameter:

```
header: pointer to the posix ACL head.
size: The space length of the posix ACL.
```

- Return value:

```
Nothing.
```

- Description:

```
This function releases the posix ACL space.
```

1.3.9 lustre_ext_acl_xattr_id2server

- Prototype:

```
int lustre_ext_acl_xattr_id2server(struct md_ucred *mu, struct lustre_idmap_table *t,
                                   ext_acl_xattr_header *header)
```

- Parameter:

```
mu: pointer to the user credential struct.
t: pointer to the user idmap table.
header: pointer to the extended ACL.
```

- Return value:

```
0: convert succeed.
-EIO: invalid extended ACL.
-EPERM: contain unmapped uid/gid.
```

- Description:

```
This function converts client-side uid/gid in the extended ACL items to server-side one
```

1.3.10 lustre_ext_acl_xattr_free

- Prototype:

```
void lustre_ext_acl_xattr_free(ext_acl_xattr_header *header)
```

- Parameter:

```
header: pointer to the extended ACL.
```

- Return value:

```
Nothing.
```

- Description:

```
This function releases the extended ACL space.
```

1.3.11 lustre_acl_xattr_merge2posix

- Prototype:

```
int lustre_acl_xattr_merge2posix(posix_acl_xattr_header *posix_header, int size,
                                 ext_acl_xattr_header *ext_header,
                                 posix_acl_xattr_header **out)
```

- Parameter:

```
posix_header: pointer to the posix ACL head.
size: size of posix ACL.
ext_header: pointer to the extended ACL.
out: double pointer to the posix ACL generated.
```

- Return value:

```
>= 0: size of the generated posix ACL.
-ENOMEM: not enough memory.
-EIO: invalid extended ACL.
```

- Description:

```
This function merges the posix ACL and the extended ACL into new posix ACL.
```

1.3.12 lustre_acl_xattr_merge2ext

- Prototype:

```
ext_acl_xattr_header *lustre_acl_xattr_merge2ext(posix_acl_xattr_header *posix_header,
                                                 int size,
                                                 ext_acl_xattr_header *ext_header)
```

- Parameter:

```
posix_header: pointer to the posix ACL head.
size: size of posix ACL.
ext_header: pointer to the extended ACL.
```

- Return value:

```
valid pointer: pointer to the extended ACL generated.
-ENOMEM: not enough memory.
-EIO: invalid ACL.
```

- Description:

```
This function merges the posix ACL and the extended ACL into new extended ACL.
```

1.4 LLITE process for remote_acl

1.4.1 Define rmtacl_ctl related struct

LLITE kernel records lustre mountpoint ioctl input as "`rmtacl_ctl_entry`" linked into "`rmtacl_ctl_table`" which is stored in the "`ll_sb_info`":

```
enum {
        RCE_HASHES      = 32
};
struct rmtacl_ctl_entry {
        struct list_head rce_list;
        pid_t            rce_key; /* hash key */
        int              rce_ops; /* acl operation type */
};
struct rmtacl_ctl_table {
        spinlock_t       rct_lock;
        struct list_head rct_entries[RCE_HASHES];
};
struct ll_sb_info {
    struct list_head        ll_list;
    ...
    struct rmtacl_ctl_table ll_rct;
};
```

1.4.2 rct_init

- Prototype:

  `void rct_init(struct rmtacl_ctl_table *rct)`

- Parameter:

  `rct: pointer to the rmtacl_ctl_table, generally, it is &ll_sb_info.ll_rct.`

- Return value:

  `Nothing.`

- Description:

  `This function initializes the rmtacl_ctl_table.`

1.4.3 rct_fini

- Prototype:

  `void rct_fini(struct rmtacl_ctl_table *rct)`

- Parameter:

  `rct: pointer to the rmtacl_ctl_table, generally, it is &ll_sb_info.ll_rct.`

- Return value:

Nothing.

- Description:

This function cleanups the rmtacl_ctl_table, including releases all the rmtacl_ctl_entr

1.4.4 rct_search_locked

- Prototype:

```
struct rmtacl_ctl_entry *rct_search_locked(struct rmtacl_ctl_table *rct, pid_t key)
```

- Parameter:

rct: pointer to the rmtacl_ctl_table.
key: for matching rmtacl_ctl_entry.rce_key.

- Return value:

NULL: can not find any rmtacl_ctl_entry with the given key.
non-NULL: pointer to the rmtacl_ctl_entry with the given key.

- Description:

This function searches the rmtacl_ctl_table to find the related rmtacl_ctl_entry with t

1.4.5 rct_add

- Prototype:

```
int rct_add(struct rmtacl_ctl_table *rct, pid_t key, int ops)
```

- Parameter:

rct: pointer to the rmtacl_ctl_table.
key: for matching rmtacl_ctl_entry.rce_key.
ops: operation types as defined in 1.1.

- Return value:

0: succeed.
-ENOMEM: not enough memory.

- Description:

This function creates new rmtacl_ctl_entry with the given key and inserts it into the r

1.4.6 rct_del

- Prototype:

  ```
  int rct_del(struct rmtacl_ctl_table *rct, pid_t key)
  ```

- Parameter:

  ```
  rct: pointer to the rmtacl_ctl_table.
  key: for matching rmtacl_ctl_entry.rce_key.
  ```

- Return value:

  ```
  0: succeed.
  -ENOENT: can not find any related rmtacl_ctl_entry with the given key.
  ```

- Description:

  ```
  This function searches the rmtacl_ctl_entry with the given key, removes it from the rmt
  ```

1.4.7 Define ext_xattr_acl related struct

For "lfs lsetfacl", LLITE kernel backups the posix ACL fetched by "getxattr" from MDS as "ext_xattr_acl" linked into "eacl_table" which is stored in the "`ll_sb_info`":

```
enum {
        EE_HASHES       = 32
};
struct eacl_entry {
        struct list_head      ee_list;
        pid_t                 ee_key; /* hash key */
        struct lu_fid         ee_fid;  /* file owner of the eacl */
        int                   ee_type; /* ACL type for ACCESS or DEFAULT */
        ext_acl_xattr_header *ee_acl;
};
struct eacl_table {
        spinlock_t       et_lock;
        struct list_head et_entries[EE_HASHES];
};
struct ll_sb_info {
    struct list_head        ll_list;
    ...
    struct rmtacl_ctl_table ll_rct;
    struct eacl_table       ll_et;
};
```

1.4.8 et_init

- Prototype:

  ```
  void et_init(struct eacl_table *et)
  ```

- Parameter:

  ```
  et: pointer to the eacl_table, generally, it is &ll_sb_info.ll_et.
  ```

- Return value:

  ```
  Nothing.
  ```

- Description:

  ```
  This function initializes the eacl_table.
  ```

1.4.9 et_fini

- Prototype:

  ```
  void et_fini(struct eacl_table *et)
  ```

- Parameter:

  ```
  et: pointer to the eacl_table, generally, it is &ll_sb_info.ll_et.
  ```

- Return value:

  ```
  Nothing.
  ```

- Description:

  ```
  This function cleanups the eacl_table, including releases all the eacl_entry.
  ```

1.4.10 ee_add

- Prototype:

  ```
  int ee_add(struct eacl_table *et, pid_t key, struct lu_fid *fid, int type,
             ext_acl_xattr_header *header)
  ```

- Parameter:

```
et: pointer to the eacl_table.
key: for matching eacl_entry.ee_key.
fid: which object the extended ACL belongs to.
type: ACL type for ACCESS or DEFAULT.
header: pointer to the extended ACL.
```

- Return value:

```
0: succeed.
-ENOMEM: not enough memory.
```

- Description:

```
This function creates new eacl_entry with the given parameters and inserts it into the
```

1.4.11 ee_search_del_locked

- Prototype:

```
struct eacl_entry *et_search_del_locked(struct eacl_table *et, pid_t key,
                                        struct lu_fid *fid, int type)
```

- Parameter:

```
et: pointer to the eacl_table.
key: for matching eacl_entry.ee_key.
fid: which object the extended ACL belongs to.
type: ACL type for ACCESS or DEFAULT.
```

- Return value:

```
NULL: can not find any eacl_entry with the given parameters.
non-NULL: pointer to the eacl_entry with the given parameters.
```

- Description:

```
This function searches the eacl_table to find the related eacl_entry with the given par
```

1.5 MDT process for remote_acl

1.5.1 mdt_rmtlsetfacl

- Prototype:

```
int mdt_rmtlsetfacl(struct mdt_thread_info *info, char *xattr_name,
                    ext_acl_xattr_header *header, void **out)
```

- Parameter:

```
info: pointer to mdt_thread_info.
xattr_name: ACL type for ACCESS or DEFAULT.
header: pointer to the extended ACL.
out: double pointer to the posix ACL generated.
```

- Return value:

```
>= 0: size of the generated posix ACL.
-ENOMEM: not enough memory.
-EIO: invalid extended ACL.
```

- Description:

```
This function converts client-side uid/gid in the extended ACL to server-side ones, and
```

2 Use Cases

2.1 posix ACL test

Lustre sanity test has contained posix ACL test for local_acl already. We can replace all the "{s,g}etfacl" with "lfs l{s,g}etfacl" to generate new posix ACL test for remote_acl.

2.2 functional test

It has been described clearly in the HLD: <**High Level Design of v2 for Remote ACL**> chapter 4 (Use Cases).

3 Logic Specification

3.1 User level process

3.1.1 rgetfacl_output

```
/* get next division or the contrary */
static char *next_token(char *p, int div)
{
    if (p == NULL)
        return NULL;
    if (div)
```

```
        while (*p && (*p != ':') && !isspace(*p))
            p++;
    else
        while ((*p == ':') || isspace(*p))
            p++;
    return *p ? p : NULL;
}
...
typedef struct {
    char *name;        /* user/group name */
    int   length;      /* name length */
    int   is_user;     /* is user name or not */
    int   next_token; /* whether need to call "next_token()" */
} rmtacl_name_t;
#define RMTACL_OPTNAME(name) name, sizeof(name) - 1
static rmtacl_name_t rmtacl_namelist[] = {
        { RMTACL_OPTNAME("user:"),            1,      0 },
        { RMTACL_OPTNAME("group:"),           0,      0 },
        { RMTACL_OPTNAME("default:user:"),    1,      0 },
        { RMTACL_OPTNAME("default:group:"),   0,      0 },
        /* for --tabular option */
        { RMTACL_OPTNAME("user"),             1,      1 },
        { RMTACL_OPTNAME("group"),            0,      1 },
        { 0 }
};
...
static int rgetfacl_output(char *str)
{
    char *start = NULL, *end = NULL;
    int is_user = 0, n, id;
    char c;
    rmtacl_name_t *rn;
    if (str == NULL)
        return -1;
    /* search name in the @str */
    for (rn = rmtacl_namelist; rn->name; rn++) {
        if(strncmp(str, rn->name, rn->length) == 0) {
            if (!rn->next_token)
                start = str + rn->length;
            else
                start = next_token(str + rn->length, 0);
            is_user = rn->is_user;
            break;
        }
    }
    end = next_token(start, 1);
```

```
        if ((end == NULL) || (start == end)) {
            n = printf("%s", str);
            return n;
        }
        c = *end;
        *end = 0;
        /* convert name to id based on the client-side user database */
        id = rmtacl_name2id(start, is_user);
        if (id == INVALID_ID) {
            if (str_is_id(start)) {
                *end = c;
                n = printf("%s", str);
            } else
               return -1;
       } else if ((id == NOBODY_UID && is_user) || (id == NOBODY_GID && !is_user)) {
            /* ignore "nobody" case */
            *end = c;
            n = printf("%s", str);
       } else {
            *end = c;
            *start = 0;
            n = printf("%s%d%s", str, id, end);
       }
       return n;
    }
```

3.1.2 do_rmtacl

```
    static int rmtacl_notify(int ops)
    {
        int fd;
        for each lustre mountpoint {
            fd = open(lustre mountpoint);
            ioctl(fd, LL_IOC_RMTACL, ops);
            /*do NOT close the fd, it will be closed
           *automatically when process exit. */
        }
        return 0;
    }
    ...
    static int do_rmtacl(int argc, char *argv[], int ops, int (output_func)(char *))
    {
        int fd[2];
        if (output_func) {
            pipe(fd);
            fork();
```

```
    }
    if (child process)
        redirect its output to fd[1];
    if (child process || !output_func) {
        rmtacl_notify(ops);
        execvp(argv[0], argv);
    }
    if (output_func && parent process) {
        read string from fd[0];
        output_func(string);
    }
    return child process exit value;
}
```

3.2 Idmap related process in OBDCLASS

3.2.1 idmap_search_entry

```
/*
 * It is mainly called before add new idmap_entry into idmap_table.
 * Return value:
 * @NULL: not found required entry
 * @ERR_PTR(-EACCES): found "1(remote):N(local)" mapped entry
 * @others: found normal entry
 */
static struct lustre_idmap_entry *idmap_search_entry(struct lustre_idmap_table *t,
                                                     uid_t rmt_uid, uid_t lcl_uid,
                                                     gid_t rmt_gid, gid_t lcl_gid)
{
    struct list_head *head;
    struct lustre_idmap_entry *e;
    /* search uid hash table to make sure no 1:N uid mapping */
    head = &t->lit_idmaps[RMT_UIDMAP_IDX][lustre_idmap_hashfunc(rmt_uid)];
    list_for_each_entry(e, head, lie_rmt_uid_hash)
        if (e->lie_rmt_uid == rmt_uid) {
            if (e->lie_lcl_uid == lcl_uid) {
                if (e->lie_rmt_gid == rmt_gid &&
                    e->lie_lcl_gid == lcl_gid)
                    /* must be quaternion match */
                    return e;
            } else {
                /* found 1:N uid mapping, return -EACCESS to
                 * client to notify that the specified user
                 * can not access the LUSTRE. */
                ...
                return ERR_PTR(-EACCES);
```

```
                }
            }
        /* search gid hash table to make sure no 1:N gid mapping */
        head = &t->lit_idmaps[RMT_GIDMAP_IDX][lustre_idmap_hashfunc(rmt_gid)];
        list_for_each_entry(e, head, lie_rmt_gid_hash)
            if (e->lie_rmt_gid == rmt_gid) {
                if (e->lie_lcl_gid == lcl_gid) {
                    if (unlikely(e->lie_rmt_uid == rmt_uid &&
                                 e->lie_lcl_uid == lcl_uid))
                        /* after uid mapping search above,
                         * we should never come here */
                        LBUG();
                } else {
                    /* found 1:N gid mapping */
                    ...
                    return ERR_PTR(-EACCES);
                }
            }
        return NULL;
    }
```

3.2.2 lustre_idmap_lookup_uid

```
    int lustre_idmap_lookup_uid(struct md_ucred *mu, struct lustre_idmap_table *t,
                                int reverse, uid_t uid)
    {
        struct list_head *hash;
        /* search "mu" first, it is a cache of idmap table equivalently.
         * there is no such process in old "lustre_idmap_lookup_uid". */
        if (mu && (mu->mu_valid == UCRED_OLD || mu->mu_valid == UCRED_NEW)) {
            if (!reverse) {
                if (uid == mu->mu_o_uid)
                    return mu->mu_uid;
                else if (uid == mu->mu_o_fsuid)
                    return mu->mu_fsuid;
            } else if (!(mu->mu_squash & SQUASH_UID)) {
                if (uid == mu->mu_uid)
                    return mu->mu_o_uid;
                else if (uid == mu->mu_fsuid)
                    return mu->mu_o_fsuid;
            }
        }
        /* then search uid mapping hash table:
         * "t->lit_idmaps[reverse ? LCL_UIDMAP_IDX : RMT_UIDMAP_IDX]";
         * as old "lustre_idmap_lookup_uid" does. */
```

```
    ...
}
```

3.3 ACL related process in OBDCLASS

3.3.1 lustre_posix_acl_xattr_filter

```
#define CFS_ACL_XATTR_SIZE(count, prefix) \
        (sizeof(prefix ## _header) + (count) * sizeof(prefix ## _entry))
#define CFS_ACL_XATTR_COUNT(size, prefix) \
        (((size) - sizeof(prefix ## _header)) / sizeof(prefix ## _entry))
/* Filter out the "nobody" entries in the posix ACL.  */
int lustre_posix_acl_xattr_filter(posix_acl_xattr_header *header, int size,
                                  posix_acl_xattr_header **out)
{
    int count, i, j, rc = 0;
    posix_acl_xattr_header *new;

    OBD_ALLOC(new, size);
    new->a_version = cpu_to_le32(CFS_ACL_XATTR_VERSION);
    count = CFS_ACL_XATTR_COUNT(size, posix_acl_xattr);
    for (i = 0, j = 0; i < count; i++) {
        switch (header->a_entries[i].e_tag) {
            case ACL_USER_OBJ:
            case ACL_GROUP_OBJ:
            case ACL_MASK:
            case ACL_OTHER:
                if (header->a_entries[i].e_id != ACL_UNDEFINED_ID)
                     return -EIO;
            case ACL_USER:
            case ACL_GROUP:
                copy “non-nobody” entry from header->a_entries[i] to new->a_entries[j++
                break;
            default:
                return -EIO;
        }
    }
    /* free unused space. */
    size = lustre_posix_acl_xattr_reduce_space(&new, count, j);
    *out = new;
    return size;
}
```

3.3.2 lustre_posix_acl_xattr_id2client

```
/*
```

```
 * Convert server-side uid/gid in the posix ACL items to the client-side ones.
 * convert rule:
 * @CFS_IC_NOTHING
 *  nothing to be converted.
 * @CFS_IC_ALL
 *  mapped ids are converted to client-side ones,
 *  unmapped ones are converted to "nobody".
 * @CFS_IC_MAPPED
 *  only mapped ids are converted to "nobody".
 * @CFS_IC_UNMAPPED
 *  only unmapped ids are converted to "nobody".
 */
int lustre_posix_acl_xattr_id2client(struct md_ucred *mu, struct lustre_idmap_table *t,
                                     posix_acl_xattr_header *header, int size, int flag
{
    int count, i;
    __u32 id;
    count = CFS_ACL_XATTR_COUNT(size, posix_acl_xattr);
    for (i = 0; i < count; i++) {
        switch(header->a_entries[i].e_tag) {
            case ACL_USER_OBJ:
            case ACL_GROUP_OBJ:
            case ACL_MASK:
            case ACL_OTHER:
                if (header->a_entries[i].e_id != ACL_UNDEFINED_ID)
                    return -EIO;
                break;
            case ACL_USER:
                id = lustre_idmap_lookup_uid(mu, t, 1, header->a_entries[i].e_id);
                if (flags == CFS_IC_ALL) {
                    /* mapped ids are converted to client-side ones,
                     * unmapped ones are converted to ‘‘nobody’’. */
                    if (id == CFS_IDMAP_NOTFOUND)
                        header->a_entries[i].e_id = NOBODY_UID;
                    else
                        header->a_entries[i].e_id = id;
                } else if (flags == CFS_IC_MAPPED) {
                    /* only mapped ids are converted to ‘‘nobody’’. */
                    if (id != CFS_IDMAP_NOTFOUND)
                        header->a_entries[i].e_id = NOBODY_UID;
                } else if (flags == CFS_IC_UNMAPPED) {
                    /* only unmapped ids are converted to ‘‘nobody’’. */
                    if (id == CFS_IDMAP_NOTFOUND)
                        header->a_entries[i].e_id = NOBODY_UID;
                }
                break;
```

```
                case ACL_GROUP:
                    id = lustre_idmap_lookup_gid(mu, t, 1, header->a_entries[i].e_id);
                    if (flags == CFS_IC_ALL) {
                        if (id == CFS_IDMAP_NOTFOUND)
                            header->a_entries[i].e_id = NOBODY_GID;
                        else
                            header->a_entries[i].e_id = id;
                    } else if (flags == CFS_IC_MAPPED) {
                        if (id != CFS_IDMAP_NOTFOUND)
                            header->a_entries[i].e_id = NOBODY_GID;
                    } else if (flags == CFS_IC_UNMAPPED) {
                        if (id == CFS_IDMAP_NOTFOUND)
                            header->a_entries[i].e_id = NOBODY_GID;
                    }
                    break;
                default:
                    return -EIO;
            }
        }
        return 0;
    }
```

3.3.3 lustre_ext_acl_xattr_id2server

```
    /*
     * Converts client-side uid/gid in the extended ACL items to server-side ones.
     * convert rule:
     *  mapped ids are converted to server-side ones,
     *  unmapped ones cause "EPERM" error.
     */
    int lustre_ext_acl_xattr_id2server(struct md_ucred *mu, struct lustre_idmap_table *t,
                                       ext_acl_xattr_header *header)
    {
        int i;
        __u32 id;
        for (i = 0; i < header->a_count; i++) {
            switch(header->a_entries[i].e_tag) {
                case ACL_USER_OBJ:
                case ACL_GROUP_OBJ:
                case ACL_MASK:
                case ACL_OTHER:
                    if (header->a_entries[i].e_id != ACL_UNDEFINED_ID)
                        return -EIO;
                    break;
                case ACL_USER:
                    id = lustre_idmap_lookup_uid(mu, t, 0, header->a_entries[i].e_id);
```

```
                if (id == CFS_IDMAP_NOTFOUND)
                    return -EPERM;
                else
                    header->a_entries[i].e_id = id;
                break;
            case ACL_GROUP:
                id = lustre_idmap_lookup_gid(mu, t, 0, header->a_entries[i].e_id);
                if (id == CFS_IDMAP_NOTFOUND)
                    return -EPERM;
                else
                    header->a_entries[i].e_id = id;
                break;
          default:
                return -EIO;
        }
    }
    return 0;
}
```

3.3.4 lustre_ext_acl_xattr_search

```
static ext_acl_xattr_entry *lustre_ext_acl_xattr_search(ext_acl_xattr_header *header,
                                                        posix_acl_xattr_entry *entry,
                                                        int *pos)
{
    int i, j, start = *pos, end = header->a_count, once = 0;
again:
    for (i = start; i < end; i++) {
        if (header->a_entries[i].e_tag == entry->e_tag &&
            header->a_entries[i].e_id == entry->e_id) {
            j = i;
            if (++i >= header->a_count)
                i = 0;
            *pos = i;
            return &header->a_entries[j];
        }
    }
    if (!once) {
        once = 1;
        start = 0;
        end = *pos;
        goto again;
    }
    return NULL;
}
```

3.3.5 lustre_acl_xattr_merge2posix

```
/*
 * Merge the posix ACL and the extended ACL into new posix ACL.
 */
int lustre_acl_xattr_merge2posix(posix_acl_xattr_header *posix_header, int size,
                                 ext_acl_xattr_header *ext_header,
                                 posix_acl_xattr_header **out)
{
    int posix_count, posix_size, i, j, pos = 0, rc = 0;
    posix_acl_xattr_entry pe = {ACL_MASK, 0, ACL_UNDEFINED_ID};
    posix_acl_xattr_header *new;
    ext_acl_xattr_entry *ee;
    ee = lustre_ext_acl_xattr_search(ext_header, &pe, &pos);
    if (ee == NULL || ee->e_stat == ES_DEL) {
        /* there are only base ACL entries at most. */
        posix_count = 3;
        posix_size = CFS_ACL_XATTR_SIZE(posix_count, posix_acl_xattr);
        OBD_ALLOC(new, posix_size);
        new->a_version = cpu_to_le32(CFS_ACL_XATTR_VERSION);
        for (i = 0, j = 0; i < ext_header->a_count; i++) {
            switch(ext_header->a_entries[i].e_tag) {
                case ACL_USER_OBJ:
                case ACL_GROUP_OBJ:
                case ACL_OTHER_OBJ:
                    if (ext_header->a_entries[i].e_id != ACL_UNDEFINED_ID)
                        return -EIO;
                    if (ext_header->a_entries[i].e_stat != ES_DEL) {
                        new->a_entries[j].e_tag = ext_header->a_entries[i].e_tag;
                        new->a_entries[j].e_perm = ext_header->a_entries[i].e_perm;
                        new->a_entries[j++].e_id = ext_header->a_entries[i].e_id;
                    }
                    break;
                case ACL_USER:
                case ACL_GROUP:
                    if (ext_header->a_entries[i].e_stat != ES_DEL)
                        return -EIO;
                    break;
                default:
                    return -EIO;
            }
        }
    } else {
        /* maybe there are valid ACL_USER or ACL_GROUP entries in the
         * original server-side ACL, they are regarded as ES_UNC stat. */
        int ori_posix_count;
```

```
            ori_posix_count= CFS_ACL_XATTR_COUNT(size, posix_acl_xattr);
            posix_count = ori_posix_count + ext_header->a_count;
            posix_size = CFS_ACL_XATTR_SIZE(posix_count, posix_acl_xattr);
            OBD_ALLOC(new, posix_size);
            new->a_version = cpu_to_le32(CFS_ACL_XATTR_VERSION);
            /* 1. process the unchanged ACL entries
             * in the original server-side ACL. */
            pos = 0;
            for (i = 0, j = 0; i < ori_posix_count; i++) {
                ee = lustre_ext_acl_xattr_search(ext_header,
                                                 &posix_header->a_entries[i], &pos);
                if (ee == NULL)
                    copy posix_header->a_entries[i] to new->a_entries[j++];
            }
            /* 2. process the non-deleted entries
             * from client-side extended ACL. */
            for (i = 0; i < ext_header->a_count; i++) {
                if (ext_header->a_entries[i].e_stat != ES_DEL) {
                    new->a_entries[j].e_tag = ext_header->a_entries[i].e_tag;
                    new->a_entries[j].e_perm = ext_header->a_entries[i].e_perm;
                    new->a_entries[j++].e_id = ext_header->a_entries[i].e_id;
                }
            }
        }
        /* free the unused space. */
        posix_size = lustre_posix_acl_xattr_reduce_space(&new, posix_count, j);
        *out = new;
        return posix_size;
    }
```

3.3.6 lustre_acl_xattr_merge2ext

```
    /*
     * Merge the posix ACL and the extended ACL into new extended ACL.
     */
    ext_acl_xattr_header *lustre_acl_xattr_merge2ext(posix_acl_xattr_header *posix_header,
                                                     int size,
                                                     ext_acl_xattr_header *ext_header)
    {
        int ori_ext_count, posix_count, ext_count, ext_size;
        int i, j, pos = 0, rc = 0;
        ext_acl_xattr_entry *ee;
        ext_acl_xattr_header *new;
        posix_count = CFS_ACL_XATTR_COUNT(size, posix_acl_xattr);
        ext_count = posix_count + ext_header->a_count;
        ext_size = CFS_ACL_XATTR_SIZE(ext_count, ext_acl_xattr);
```

```
OBD_ALLOC(new, ext_size);
for (i = 0, j = 0; i < posix_count; i++) {
    switch (posix_header->a_entries[i].e_tag) {
        case ACL_USER_OBJ:
        case ACL_GROUP_OBJ:
        case ACL_MASK:
        case ACL_OTHER:
            if (posix_header->a_entries[i].e_id != ACL_UNDEFINED_ID)
                GOTO(out, rc = -EIO);
        case ACL_USER:
            /* ignore "nobody" entry. */
            if (posix_header->a_entries[i].e_id == NOBODY_UID)
                break;
        case ACL_GROUP:
            /* ignore "nobody" entry. */
            if (posix_header->a_entries[i].e_id == NOBODY_GID)
                break;
            new->a_entries[j].e_tag = posix_header->a_entries[i].e_tag;
            new->a_entries[j].e_perm = posix_header->a_entries[i].e_perm;
            new->a_entries[j].e_id = posix_header->a_entries[i].e_id;
            ee = lustre_ext_acl_xattr_search(ext_header,
                                             &posix_header->a_entries[i], &pos);
            if (ee) {
                if (posix_header->a_entries[i].e_perm != ee->e_perm)
                    /* entry modified. */
                    ee->e_perm = new->a_entries[j++].e_stat = ES_MOD;
                else
                    /* entry unchanged. */
                    ee->e_perm = new->a_entries[j++].e_stat = ES_UNC;
            } else {
                /* new entry. */
                new->a_entries[j++].e_stat = ES_ADD;
            }
            break;
        default:
            return -EIO;
    }
}
/* process deleted entries. */
for (i = 0; i < ext_header->a_count; i++) {
    if (ext_header->a_entries[i].e_stat == ES_UNK) {
        /* ignore "nobody" entry. */
        if ((ext_header->a_entries[i].e_tag == ACL_USER &&
            ext_header->a_entries[i].e_id == NOBODY_UID) ||
            (ext_header->a_entries[i].e_tag == ACL_GROUP &&
            ext_header->a_entries[i].e_id == NOBODY_GID))
```

```
                continue;
            new->a_entries[j].e_tag = ext_header->a_entries[i].e_tag;
            new->a_entries[j].e_perm = ext_header->a_entries[i].e_perm;
            new->a_entries[j].e_id = ext_header->a_entries[i].e_id;
            new->a_entries[j++].e_stat = ES_DEL;
        }
    }
    new->a_count = cpu_to_le32(j);
    /* free unused space. if "j == 0", then "new = {0, NULL}", NOT NULL. */
    rc = lustre_ext_acl_xattr_reduce_space(&new, ext_count);
    return new;
}
```

3.4 LLITE process

3.4.1 ll_dir_ioctl

```
static int ll_dir_ioctl(struct inode *inode, struct file *file,
                        unsigned int cmd, unsigned long arg)
{
    ...
    switch(cmd) {
        ...
#ifdef CONFIG_FS_POSIX_ACL
        /* for remote_acl, insert rmtacl_ctl_entry into rmtacl_ctl_table,
         * and set ‘‘LL_FILE_RMTACL’’ to lustre mountpoint fd->fd_flags. */
        case LL_IOC_RMTACL: {
            if (sbi->ll_flags & LL_SBI_RMT_CLIENT &&
                inode == inode->i_sb->s_root->d_inode) {
                struct ll_file_data *fd = LUSTRE_FPRIVATE(file);
                rc = rct_add(&sbi->ll_rct, cfs_curproc_pid(), arg);
                if (!rc)
                        fd->fd_flags |= LL_FILE_RMTACL;
                RETURN(rc);
            } else
                RETURN(0);
        }
#endif
        ...
    }
    ...
}
```

3.4.2 ll_file_release

```
int ll_file_release(struct inode *inode, struct file *file)
{
    ...
    ENTRY;
    CDEBUG(D_VFSTRACE, "VFS Op:inode=%lu/%u(%p)\n", inode->i_ino,
           inode->i_generation, inode);
#ifdef CONFIG_FS_POSIX_ACL
        if (sbi->ll_flags & LL_SBI_RMT_CLIENT &&
            inode == inode->i_sb->s_root->d_inode) {
                struct ll_file_data *fd = LUSTRE_FPRIVATE(file);
                if (unlikely(fd->fd_flags & LL_FILE_RMTACL)) {
                        struct eacl_entry *ee;
                        /* drop flags "LL_FILE_RMTACL" */
                        fd->fd_flags &= ~LL_FILE_RMTACL;
                        /* process "rmtacl_ctl_table" */
                        rct_del(&sbi->ll_rct, cfs_curproc_pid());
                        /* process "eacl_table" */
                        while (1) {
                                ee = et_search_del_locked(&sbi->ll_et, cfs_curproc_pid(
                                                          NULL, 0);
                                if (ee)
                                        ee_free(ee);
                                else
                                        break;
                        }
                }
        }
#endif
    ...
}
```

3.4.3 ll_setxattr_common

```
static int ll_setxattr_common(struct inode *inode, const char *name, const void *value,
                              size_t size, int flags, __u64 valid)
{
    ...
    posix_acl_xattr_header *new_value = NULL;
    struct rmtacl_ctl_entry *rce = NULL;
    ext_acl_xattr_header *acl = NULL;
    ...
    /* b10667: ignore lustre special xattr for now */
```

```
    if (xattr_type == XATTR_TRUSTED_T && strcmp(name, "trusted.lov") == 0
        RETURN(0);
#ifdef CONFIG_FS_POSIX_ACL
    if ((sbi->ll_flags & LL_SBI_RMT_CLIENT) &&
       (xattr_type == XATTR_ACL_ACCESS_T ||
        xattr_type == XATTR_ACL_DEFAULT_T)) {
        /* obtain the rmtacl_ctl_entry. */
        rce = rct_search_locked(&sbi->ll_rct, cfs_curproc_pid());
        if (rce == NULL ||
           (rce->rce_ops != RMT_LSETFACL &&
            rce->rce_ops != RMT_RSETFACL))
            RETURN(-EOPNOTSUPP);
        if (rce->rce_ops == RMT_LSETFACL) {
            /* merge with backuped ext_ACL for "lfs lsetfacl" case. */
            struct eacl_entry *ee;
            ee = et_search_del_locked(&sbi->ll_et, cfs_curproc_pid(),
                                      ll_inode2fid(inode), xattr_type);
            if (valid & OBD_MD_FLXATTR)
                 acl = lustre_acl_xattr_merge2ext(value, size, ee->ee_acl);
        } else if (rce->rce_ops == RMT_RSETFACL)
            /* drop "nobody" entry for "lfs rsetfacl" case. */
            size = lustre_posix_acl_xattr_filter(value, size, &new_value);
        else
            RETURN(-EOPNOTSUPP);
    }
#endif
    oc = ll_mdscapa_get(inode);
    if (acl != NULL)
        rc = md_setxattr(sbi->ll_md_exp, ll_inode2fid(inode), oc,
                         valid | rce_ops2valid(rce->rce_ops),
                         name, (const char *)acl,
                         CFS_ACL_XATTR_SIZE(le32_to_cpu(acl->a_count), ext_acl_xattr),
                         0, flags, &req);
    else
        rc = md_setxattr(sbi->ll_md_exp, ll_inode2fid(inode), oc,
                         valid | (rce ? rce_ops2valid(rce->rce_ops) : 0),
                         name, new_value ? : value, size, 0, flags, &req);
    capa_put(oc);
#ifdef CONFIG_FS_POSIX_ACL
    if (new_value != NULL)
        lustre_posix_acl_xattr_free(new_value, size)
    if (acl != NULL)
        lustre_ext_acl_xattr_free(acl);
#endif
    ...
}
```

3.4.4 ll_getxattr_common

```
static int ll_getxattr_common(struct inode *inode, const char *name, void *buffer,
                              size_t size, __u64 valid)
{
    ...
    struct rmtacl_ctl_entry *rce = NULL;
    ...
    xattr_type = get_xattr_type(name);
    rc = xattr_type_filter(sbi, xattr_type);
    if (rc)
        RETURN(rc);
#ifdef CONFIG_FS_POSIX_ACL
    if ((sbi->ll_flags & LL_SBI_RMT_CLIENT) &&
       (xattr_type == XATTR_ACL_ACCESS_T ||
        xattr_type == XATTR_ACL_DEFAULT_T)) {
        /* obtain the rmtacl_ctl_entry. */
        rce = rct_search_locked(&sbi->ll_rct, cfs_curproc_pid());
        if (rce == NULL ||
           (rce->rce_ops != RMT_LSETFACL &&
            rce->rce_ops != RMT_LGETFACL &&
            rce->rce_ops != RMT_RSETFACL &&
            rce->rce_ops != RMT_RGETFACL))
                RETURN(-EOPNOTSUPP);
#endif
    ...
do_getxattr:
    oc = ll_mdscapa_get(inode);
    rc = md_getxattr(sbi->ll_md_exp, ll_inode2fid(inode), oc,
                     valid | (rce ? rce_ops2valid(rce->rce_ops) : 0),
                     name, NULL, 0, size, 0, &req);
    capa_put(oc);
    ...
    if (!xdata) {
        CERROR("can't extract: %u : %u\n", body->eadatasize,
               lustre_msg_buflen(req->rq_repmsg, REPLY_REC_OFF + 1));
        GOTO(out, rc = -EFAULT);
    }
#ifdef CONFIG_FS_POSIX_ACL
    if (body->eadatasize >= 0 && rce && rce->rce_ops == RMT_LSETFACL) {
        /* backup as ext_ACL for "lfs lsetfacl" case. */
        ext_acl_xattr_header *acl;
        acl = lustre_posix_acl_xattr_2ext((posix_acl_xattr_header *)xdata,
                                          body->eadatasize);
        rc = ee_add(&sbi->ll_et, cfs_curproc_pid(), ll_inode2fid(inode),
```

```
                    xattr_type, acl);
        }
        if (xattr_type == XATTR_ACL_ACCESS_T && !body->eadatasize)
                GOTO(out, rc = -ENODATA);
    #endif
        memcpy(buffer, xdata, body->eadatasize);
        rc = body->eadatasize;
        ...
    }
```

3.5 MDT process

3.5.1 mdt_rmtlsetfacl

```
static int mdt_rmtlsetfacl(struct mdt_thread_info *info, char *xattr_name,
                           ext_acl_xattr_header *header,
                           posix_acl_xattr_header **out)
{
        struct ptlrpc_request *req = mdt_info_req(info);
        struct mdt_export_data *med = mdt_req2med(req);
        struct md_ucred *uc = mdt_ucred(info);
        struct md_object *next = mdt_object_child(info->mti_object);
        struct lu_buf *buf = &info->mti_buf;
        int rc;
        /* convert client-side uid/gid to server-side ones. */
        rc = lustre_ext_acl_xattr_id2server(uc, med->med_idmap, header);
        rc = mo_xattr_get(info->mti_env, next, &LU_BUF_NULL, xattr_name);
        if (rc == -ENODATA)
                rc = 0;
        else if (rc < 0)
                return rc;
        buf->lb_len = rc;
        if (buf->lb_len > 0)
                OBD_ALLOC(buf->lb_buf, buf->lb_len);
        /* get the original posix ACL. */
        rc = mo_xattr_get(info->mti_env, next, buf, xattr_name);
        /* merge client-side ext_ACL and server-side posix ACL. */
        rc = lustre_acl_xattr_merge2posix((posix_acl_xattr_header *)(buf->lb_buf),
                                          buf->lb_len, header, out);
        if ((rc <= 0) && (buf->lb_buf != NULL))
                OBD_FREE(buf->lb_buf, buf->lb_len);
        return rc;
}
```

3.5.2 mdt_setxattr

```
int mdt_setxattr(struct mdt_thread_info *info)
{
    posix_acl_xattr_header *new_xattr = NULL;
    ...
    rc = mdt_init_ucred(info, reqbody);
    if (rc)
        RETURN(err_serious(rc));
    /* check admin configure for RMTACL_PERM. */
    if (med->med_rmtclient && (valid & OBD_MD_FLRMTRSETFACL)) {
        lnet_nid_t peernid = req->rq_peer.nid;
        __u32 perm = mdt_identity_get_perm(mu->mu_identity, med->med_rmtclient, peernid
        if (!(perm & CFS_RMTACL_PERM))
           GOTO(out, rc = err_serious(-EPERM));
    }
    rc = req_capsule_pack(pill);
    if (rc < 0)
        GOTO(out, rc = err_serious(rc));
    ...
    if (valid & OBD_MD_FLXATTR) {
        ...
        if (xattr_len) {
            xattr = req_capsule_client_get(pill, &RMF_EADATA);
            /* only "lfs lsetfacl" need special process. */
            if (valid & OBD_MD_FLRMTLSETFACL) {
                LASSERT(med->med_rmtclient);
                xattr_len = mdt_rmtlsetfacl(info, xattr_name,
                                            (ext_acl_xattr_header *)xattr, &new_xattr);
                if (xattr_len < 0)
                     GOTO(out_unlock, rc = xattr_len);
                xattr = (char *)new_xattr;
            }
            if (body->flags & XATTR_REPLACE)
                flags |= LU_XATTR_REPLACE;
            ...
        }
    } else if (valid & OBD_MD_FLXATTRRM) {
    ...
out_unlock:
    if (unlikely(new_xattr != NULL))
        lustre_posix_acl_xattr_free(new_xattr, xattr_len);
    ...
}
```

3.5.3 mdt_getxattr

```
int mdt_getxattr(struct mdt_thread_info *info)
{
    ...
    easize = mdt_getxattr_pack_reply(info);
    if (easize < 0)
        GOTO(out, rc = err_serious(easize));
    next = mdt_object_child(info->mti_object);
    if (med->med_rmtclient && (valid & OBD_MD_FLRMTRGETFACL)) {
        lnet_nid_t peernid = req->rq_peer.nid;
        __u32 perm = mdt_identity_get_perm(mu->mu_identity, med->med_rmtclient, peernid
        /* check admin configure for RMTACL_PERM. */
        if (!(perm & CFS_RMTACL_PERM))
            GOTO(out, rc = err_serious(-EPERM));
       /* only the file owner can do "lfs rgetfacl". */
       rc = mo_permission(info->mti_env, NULL, next, NULL, MAY_RGETFACL);
            GOTO(out, rc = err_serious(rc));
    }
    ...
    if (valid & OBD_MD_FLXATTR) {
        int flags = CFS_IC_NOTHING;
        char *xattr_name = req_capsule_client_get(&info->mti_pill, &RMF_NAME);
        CDEBUG(D_INODE, "getxattr %s\n", xattr_name);
        rc = mo_xattr_get(info->mti_env, next, buf, xattr_name);
        if (rc < 0)
            CERROR("getxattr failed: %d\n", rc);
        /* id convert before send result to client. */
        if (valid & (OBD_MD_FLRMTLSETFACL | OBD_MD_FLRMTLGETFACL))
            flags = CFS_IC_ALL;
        else if (valid & OBD_MD_FLRMTRGETFACL)
            flags = CFS_IC_MAPPED;
        if ((rc > 0) && med->med_rmtclient && (flags != CFS_IC_NOTHING)) {
            int rc1;
            rc1 = lustre_posix_acl_xattr_id2client(uc, med->med_idmap,
                                                   (posix_acl_xattr_header *)(buf->lb_bu
                                                   rc, flags);
            if (unlikely(rc1 < 0))
                rc = rc1;
        }
    } else if (valid & OBD_MD_FLXATTRLS) {
    ...
}
```

4 State Specification

4.1 Resources Involved and Their State

LUSTRE IDMAP adjustment ("N:1" mapping) should be done first, it is the base of remote_acl.

4.2 Locking

4.2.1 rct_lock

```
struct rmtacl_ctl_table {
    spinlock_t rct_lock;
    struct list_head rct_entries[RCE_HASHES];
};
```

Each LLITE super_block has only one rmtacl_ctl_table, "rct_lock" is used for controlling multiple client threads accessing (search/add/del) the rmtacl_ctl_table.

4.2.2 et_lock

```
struct eacl_table {
    spinlock_t et_lock;
    struct list_head et_entries[EE_HASHES];
};
```

Each LLITE super_block has only one eacl_table, "et_lock" is used for controlling multiple client threads accessing (search/add/del) the eacl_table.

4.2.3 lit_lock

```
struct lustre_idmap_table {
    spinlock_t lit_lock;
    struct list_head lit_idmaps[CFS_IDMAP_N_HASHES] [CFS_IDMAP_HASHSIZE];
};
```

Each MDS has only one lustre_idmap_table, "lit_lock" is used for controlling multiple server threads accessing (lookup/add/del) the lustre_idmap_table.

4.3 Recovery

Only one recovery related work must be considered: how to reestablish the "client-side uid/gid <=> server-side uid/gid" mapping which existed before MDS crashed in recovery. Without such reestablishment, remote_stat/remote_getfacl maybe get unexpected user/group: "nobody", and remote_chgrp/remote_setfacl maybe failed for unexpected error: "EPERM, unmapped user/group". So it is not a new issue addressed by the new remote_acl design.

4.4 Others

For remote user, executing system "s{g}etfacl" utils directly will cause error of "-EOPNOTSUPP". Some other system utils related ACL operation have the similar issue, e.g "ls -l", "cp -p", and so on. They all don't work as expected as executed by local user. Although it is not a new issue addressed by the new remote_acl design, maybe we need more work to resolve other ACL related issues for remote user in future. In this design, we would like to provide some new lfs utils as "lfs ls", "lfs cp" to resolve the issues temporary.

5 Environment

5.1 Network Protocol Compatibility

5.1.1 New clients and Old Servers

The feature of "remote client" will not be supported until lustre 1.8. The client type (local or remote) is determined by MDS when client connects to MDS. For old server, it ignores client type, all clients are regarded as local ones. So even if the new client declared that it was a remote client, it would be granted as local one by server yet. So in such case, no remote client, nor remote_acl.

5.1.2 Old clients and New Servers

If old client does not support "remote client", but it just belong to a remote domain, then when such client connect to a new MDS (support "remote client" feature), the MDS should refuse its connection for the whole system security.

5.2 Documentation Changes

5.2.1 Configure change

Rename server configure file "/etc/lustre/setxid.conf" to "/etc/lustre/perm.conf". Add permission flags "rmtacl/normtacl", for enable/disable permission of "lfs r{s,g}etfacl".

5.2.2 "lfs {l,r}{s,g}etfacl" utils usage

"lfs {l,r}{s,g}etfacl" utils follows the same parameter format as the system "{s,g}etfacl" utils, just as following. Please refer to man page (1) {s,g}etfacl for the detail.

```
lfs {l,r}setfacl [-bkndRLPvh] [{-m|-x} acl_spec] [{-M|-X} acl_file] file ...
lfs {l,r}setfacl --restore=file
lfs {l,g}getfacl [-dRLPvh] file ...
lfs {l,g}getfacl [-dRLPvh] -
```