# CMD Rename Locking

Yury Umanets

13th August 2006

## 1 Functional Specification

This DLD is intended to describe how should rename locking be done in CMD3. It should solve the following issues:

- introduce CMD rename sanity checks, so that source and destination may be verified if they may proceed with rename even if they are located on different MDTs;
- introduce semantically proper rename locking as described in Linux documentation. This includes protecting tree (hierarchy) from being changed while rename is proceeding, even if source and destination are located in different MDTs. This is mostly needed for cross-dir renames.

### 1.1 Sanity checks

Sanity checks should include the following:

- all usual rename sanity checks which may be done locally. No need to describe them here in details;
- check that source is not subdir of target.

### 1.2 Locking

Locking should include the following:

- usual parent-first locking as CMD3 does, this is not going to be described in more details. No need for that;
- lock whole filesystem tree:
  - to serialize cross-dir renames and avoid deadlocks in case of few concurrent renames;
  - to prevent change the filesystem parent-child order, when sanity checks run (check for subdir mostly).

What is said above means mostly two issues which this DLD should solve:

- we have to check if src directory is not subdir of dst one, like it is done in Linux is_subdir() function. This should work nicely even for CMD cases, what means, that during analizing path and walking it, each itme of path may be located on another node in cluster. That is we should have some function which will be able to talk to other MDTs in cluster (if needed) to check if directory entries stored on those nodes are not children of target directory;
- we have to serialize cross-dir and cross-node renames to avoid deadlock caused by few concurrent rename threads. To implement this we need to use DLM lock with special resource id (big filesystem lock, that is BFL). Each node starting rename should issue BFL to make sure that no other renames are proceeding.

# 2 Use Cases

**Source can't be subdir of destination**

The following test for both single mds and CMD configurations should not pass.

```
mkdir -p /mnt/lustre/a/b/c/d
mv /mnt/lustre/a/b /mnt/lustre/a/b/c/d
```

# 3 Logic Specification

## 3.1 Locking

This chapter describes locking implementation details for for most hairy cases and how is BFL implemented.

### 3.1.1 BFL basics

To implement BFL, we want to use specil resource id which is based on special FID. FIDs management saves 10M FIDs for such a special purposes. Thus, we can use the following FID for rename purposes:

```
0x0000000000000003/0x0000000000000001/0x0000000000000000
```

What means sequence number 0x3, object number 0x1, version 0x0. This FID is going to defined in fid module and declared in headers to become one of so called "well known fids".

In file lustre/fid/fid_lib.c it may look like the following:

```
struct lu_fid LUSTRE_BFL_FID = { .f_seq = 0x0000000000000003,
                                 .f_oid = 0x0000000000000001,
                                 .f_ver = 0x0000000000000000 };
EXPORT_SYMBOL(LUSTRE_BFL_FID);
```

In file lustre/include/lustre_fid.h it may look like the following:

```
extern struct lu_fid LUSTRE_BFL_FID;
```

So that, each module on client and server can use it for creating BFL lock resource id like the following:

```
struct ldlm_res_id bfl_res_id;
fid_build_res_name(&LUSTRE_BFL_FID, &bfl_res_id);
```

### 3.1.2 Locking logics

The main question here is where BFL should be taken. It may be done on MDT server that is performing rename (owner of parent of src directory) or on client executing rename operation.

One more question is where should BFL be requested, locks live in some namespace. And answers about locking logics are the following:

- BFL request should be sent to MDT0, which is cluster-controller node in terms of sequence management and also may be used for rename in such a role. That is all BFL locks should be taken in MDT0 namespace;
- BFL should be taken on MDT server, owner of parent of source in rename operation. This is because, only in this case we can be sure that FS hierarchy is not changing in same time by concurrent rename which already took all locks and will not be blocked on BFL.

The following is prototype of functions, which take and release rename lock (BFL) on MDT:

```
static int
mdt_rename_lock(struct mdt_thread_info *info,
                struct lustre_handle *lh)
{
        ldlm_policy_data_t policy =
            { .l_inodebits = {MDS_INODELOCK_UPDATE } };
        struct ldlm_namespace *ns = info->mti_mdt->mdt_namespace;
        int flags = LDLM_FL_ATOMIC_CB;
        struct ldlm_res_id res_id;
        struct lu_site *ls;
        int rc;
        ENTRY;
```

```
        ls = info->mti_mdt->mdt_md_dev.md_lu_dev.ld_site;
        fid_build_res_name(&LUSTRE_BFL_FID, &res_id);
        if (ls->ls_control_exp == NULL) {
                /*
                 * Current node is controller, that is mdt0
                 * where we should take BFL lock.
                 */
                rc = ldlm_cli_enqueue_local(ns, res_id, LDLM_IBITS,
                                            &policy, LCK_EX, &flags,
                                            ldlm_blocking_ast,
                                            ldlm_completion_ast, NULL,
                                            NULL, 0, NULL, lh);
        } else {
                /*
                 * This is the case mdt0 is remote node, issue DLM
                 * lock like other clients.
                 */
                rc = ldlm_cli_enqueue(ls->ls_control_exp, NULL, res_id,
                                      LDLM_IBITS, &policy, LCK_EX, &flags,
                                      ldlm_blocking_ast, ldlm_completion_ast,
                                      NULL, NULL, NULL, 0, NULL, lh, 0);
        }
        RETURN(rc);
}

static void mdt_rename_unlock(struct lustre_handle *lh)
{
        ENTRY;
        ldlm_lock_decref(lh, LCK_EX);
        EXIT;
}
```

BFL is taken before doing anything else in rename time and released after all rename operations are done.

## 3.2 Sanity checks

After BFL is taken, we can be sure that FS hierarchy is not going to change and may perform CMD variant of *is_subdir()* check.

It is implemented the following way:

- MDT node, owner of destination parent, calls *mdt_rename_check()* function using rename operation arguments. This function, in loop calls *mdo_subdir()* on child object to check for subdir constraint. It passes @dst child object and @src fid to *mdo_subdir()*;

- local *mdo_subdir()* passes control to cmm module, which in turn forwards it either to mdd (local @dst dir) or to mdc (remote @dst dir);
- in local case, *mdo_subdir()* is forwarded to mdd, it checks if passed @src fid is subdir of passed @dst dir using *mdd_is_parent()* method and retruns result to caller. Return value is like this:
  - 1 - passed @src fid is subdir of passed @dst dir;
  - 0 - passed @src fid is not a subdir of @dst, check went up to root and found nothing like this;
  - EREMOTE - local check in mdd module found remote object and upper layer should forward request to another target. In this case, *mdo_subdir()* also returns fid of last checked parent in @dst path. Returned fid is used later for continue check on remote MDT;
- in the case @dst dir is remote object, cmm and mdc will call remote *md_subdir()* method which checks the same on remote MDT. Return value is the same as in previous point;

The following is prototype of *mdt_rename_check()* function.

```
static int
mdt_rename_check(struct mdt_thread_info *info,
                 struct lu_fid *fid)
{
        struct mdt_reint_record *rr = &info->mti_rr;
        struct lu_fid dst_fid = *rr->rr_fid2;
        struct mdt_object *dst;
        int rc = 0;
        ENTRY;

        do {
                dst = mdt_object_find(info->mti_ctxt, info->mti_mdt,
                                      &dst_fid);
                if (!IS_ERR(dst)) {
                        rc = mdo_subdir(info->mti_ctxt,
                                        mdt_object_child(dst),
                                        fid, &dst_fid);
                        mdt_object_put(info->mti_ctxt, dst);
                        if (rc < 0) {
                                CERROR("Error while doing "
                                       "mdo_subdir(), rc %d\n",
                                       rc);
                        } else if (rc == 1) {
                                rc = -EINVAL;
                        }
```

```
                } else {
                        rc = PTR_ERR(dst);
                }
        } while (rc == EREMOTE);
        RETURN(rc);
}
```

The following is *mdo_subdir()* implementation in cmm module:

```
static int
cmm_subdir(const struct lu_context *ctx, struct md_object *mo,
           const struct lu_fid *fid, struct lu_fid *sfid)
{
        int rc;
        ENTRY;
        rc = mdo_subdir(ctx, md_object_next(mo), fid, sfid);
        RETURN(rc);
}
```

And this is mdd module implementation of *mdo_subdir()*:

```
static int
mdd_subdir(const struct lu_context *ctx, struct md_object *mo,
           const struct lu_fid *fid, struct lu_fid *sfid)
{
        struct mdd_device *mdd = mdo2mdd(mo);
        int rc;
        ENTRY;

        rc = mdd_is_parent(ctx, mdd, mo, fid, sfid);
        if (rc == -EREMOTE)
                rc = EREMOTE;
        RETURN(rc);
}
```

And for case of remote call, cmm child mdc object has the following *mdo_subdir()* implementation:

```
static int
mdc_subdir(const struct lu_context *ctx, struct md_object *mo,
           const struct lu_fid *fid, struct lu_fid *sfid)
{
        struct mdc_device *mc = md2mdc_dev(md_obj2dev(mo));
        struct mdc_thread_info *mci;
        struct mdt_body *body;
        int rc;
        ENTRY;
```

```
        mci = mdc_info_init(ctx);
        rc = md_subdir(mc->mc_desc.cl_exp, lu_object_fid(&mo->mo_lu),
                       fid, &mci->mci_req);
        if (rc)
                GOTO(out, rc);
        body = lustre_msg_buf(mci->mci_req->rq_repmsg, REPLY_REC_OFF,
                              sizeof(*body));
        LASSERT(body->valid & (OBD_MD_FLMODE | OBD_MD_FLID) &&
                (body->mode == 0 || body->mode == 1 ||
                 body->mode == EREMOTE));
        rc = body->mode;
        if (rc == EREMOTE) {
                CDEBUG(D_INFO, "Remote mdo_subdir(), new src "
                       DFID"\n", PFID(&body->fid1));
                *sfid = body->fid1;
        }
        EXIT;
out:
        ptlrpc_req_finished(mci->mci_req);
        return rc;
}
```

# 4 State Specification

## 4.1 Locking

Locking is described in above.

## 4.2 Recovery

No changes to recovery are expected.