CMD & 1.6.x Client Requests Changes DLD

Lai Siyao <lsy@clusterfs.com>

2007.04.23

1 Introduction

This document describes how 1.6.x client chooses different request formats to interoperate with both 1.6 and 1.8 server, besides the req_capsule will be introduced to 1.6.x and 1.8 client code. Understanding of cmd request layout is needed.

2 Requirements

- 1.6.x: release that introduces clients that are upward compatible with 1.8 md server. At the connect time 1.6.x client determines whether server supports fids (this is implemented through OBD_CONNECT_FID connect flag). If fids aren't supported (i.e., server is 1.6), 1.6.x client behaves like normal 1.6 client. If fids are supported, cmd3 formatted requests are used to talk with server (This might be a good point to introduce req_capsule interface on client, because it would allow better unification of code between 1.6 and 1.6.x modes.).
- ensure that there are some unused fields in all of the wire structs for expansion (e.g. mdt_body has none)
- ensure that all of the fields are properly 64-bit aligned (this should already be true, but good to double check, maybe automatically in "wirecheck" to check 8-byte alignment for 8-byte fields?)
- do all of a reint from JUST the information in the reint rec + some additional buffers (e.g. name, ea data), and not depend on anything in the mdt_body. make all of the mdt_rec_* structs have the same size and layout on the wire.

3 Functional specification

This falls into three parts:

- 1.6.x and 1.8 client will share the same mdc_md_ops, and all the request format differences will be handled in mdc_xxx_pack() functions.
- introduce req_capsule interface for 1.6.x and 1.8 client (both llite and mdc).
- get rid of lustre_shrink_reply(), use req_capsule instead.

3.1 global data type change

- stop using struct ll_fid in client code, replace it with struct lu_fid.(this is just a temporary handling in this phase, for the details please refer to fid_ino_map task, which is not ready yet)
- stop using mds_body, in 1.6.x client code, replace it with struct mdt_body. But we will keep the fields in the same offset as in 1.6 to keep compatibility.
- make all mdt_rec_* struct have the same size and layout, so we will use a virtual struct mdt_rec_reint to pack/unpack reint record.

3.2 1.6 requests layout

- 1.6 request formats don't contain capa field.
- for those requests which 1.6 and 1.8 share the same format, don't introduce new format.

All the special request formats for 1.6 are as follows:

```
/* 1.6 special requests format */
static const struct req_format *req_formats_16[] = {
        &RQF_MDS_GETSTATUS_16,
        &RQF_MDS_GETATTR_16,
        &RQF_MDS_GETATTR_NAME_16,
        &RQF_MDS_REINT_CREATE_16,
        &RQF_MDS_REINT_CREATE_SYM_16,
        &RQF_MDS_REINT_OPEN_16,
        &RQF_MDS_REINT_UNLINK_16,
        &RQF_MDS_REINT_LINK_16,
        &RQF_MDS_REINT_RENAME_16,
        &RQF_MDS_REINT_SETATTR_16,
        &RQF_LDLM_INTENT_GETATTR_16,
        &RQF_LDLM_INTENT_OPEN_16,
        &RQF_LDLM_INTENT_CREATE_16,
        &RQF_LDLM_INTENT_UNLINK_16,
        &RQF_MDS_GETXATTR_16,
```

```
        &RQF_MDS_SETXATTR_16,
        &RQF_MDS_SYNC_16,
        &RQF_MDS_CLOSE_16,
        &RQF_MDS_PIN_16,
        &RQF_MDS_READPAGE_16
};
```

3.3 embed req_capsule in ptlrpc_request

In 1.8 code, the req_capsule is embedded in mxx_thread_info, however for client side, there is no such thread_info yet, and request is not tightly bound to one call trace(eg, open request will be later used in replay or close), so we will embed the req_capsule in struct ptlrpc_request.

Both 1.6.x and 1.8 client code will use req_capsule interface to pack/unpack buffers from now on, and the existed 1.8 server code needs to be updated to take advantage of this too.

3.4 void req_capsule_shrink(struct req_capsule *pill, const struct req_msg_field *field, enum req_location loc, unsigned int newlen)

This function is introduced in 1.8 server code to shrink reply buffers. But herein we won't do the way like before: in 1.6 code shrink not only the buffers, but also the rq_buflen[] array if the buffer length is shrunk to 0. To the contrary, the rq_buflen[] won't be shrunk, and in the 0 length case the respective rq_buflen[offset] will be set to 0 to keep all reply buffers in fixed offset. This will help client code to unpack fields according to offset, which is by mdt_body.valid flag and mdt_body offset now.

Note for 1.6.x client code we will unpack buffers in the old way to keep compatibility.

4 Use cases

4.1 client setattr

1. client calls md_setattr(), which finally calls mdc_setattr().
2. mdc_setattr() prepares the request, and then calls mdc_setattr_pack() to pack request buffers.
3. upon pack mds/mdt_rec_setattr, once it finds it's sending request to 1.6 server, it calls mdc_setattr_pack_rec_16(), otherwise calls mdc_setattr_pack_rec() to pack this reint record.
4. if (op_data->op_flags | (MF_SOM_CHANGE) | MF_EPOCH_OPEN), the mdt_epoch is packed in request, this will only happen in communicating with 1.8 server.

5. for the remaining request buffers, the buffer is obtained via req_capsule_client_get(), and filled with proper value.

6. after request packing reply space is reversed and finally calls mdc_reint().

5 Logic specification

5.1 struct mdt_body

```
struct mdt_body {
        struct lu_fid  fid1;
        struct lu_fid  fid2;
        ...
        __u64          ioepoch;
        __u64          ino;    /* 1.6 only */
        ...
        __u32          nlink; /* #bytes to read in the case of MDS_READPAGE */
        __u32          generation; /* 1.6 only */
        ...
        __u32          max_mdsize;
        __u32          max_cookiesize;
        __u32          padding_4; /* also fix lustre_swab_mdt_body */
};
```

There are two fields ***ino*** and ***generation***, which are 1.6 only. And they will be renamed to padding_* in 1.8.

5.2 struct mdt_rec_*

```
struct mdt_rec_reint {
        __u32           rr_opcode;
        __u32           rr_fsuid;
        __u32           rr_fsgid;
        __u32           rr_cap;
        __u32           rr_suppgid1;
        __u32           rr_suppgid2;
        struct lu_fid   rr_fid1;
        struct lu_fid   rr_fid2;
        __u64           rr_size;
        __u64           rr_blocks;
        __u64           rr_mtime;
        __u64           rr_atime;
        __u64           rr_ctime;
        __u32           rr_flags;
```

```
        __u32           rr_padding_1;
        __u32           rr_padding_2;
        __u32           rr_padding_3;
        __u32           rr_padding_4;
        __u32           rr_padding_5;
};
```

The following mdt_rec_* struct will have the same size and layout of mdt_rec_reint.

```
 struct mdt_rec_setattr {
        __u32           sa_opcode;
        __u32           sa_fsuid;
        __u32           sa_fsgid;
        __u32           sa_cap;
        __u32           sa_suppgid;
        __u32           sa_mode;
        struct lu_fid   sa_fid;
        __u64           sa_valid;
        __u32           sa_uid;
        __u32           sa_gid;
        __u64           sa_size;
        __u64           sa_blocks;
        __u64           sa_mtime;
        __u64           sa_atime;
        __u64           sa_ctime;
        __u32           sa_attr_flags;
        __u32           sa_padding_1;
        __u32           sa_padding_2;
        __u32           sa_padding_3;
        __u32           sa_padding_4;
        __u32           sa_padding_5;
};

struct mdt_rec_create {
        __u32           cr_opcode;
        __u32           cr_fsuid;
        __u32           cr_fsgid;
        __u32           cr_cap;
        __u32           cr_flags; /* for use with open */
        __u32           cr_mode;
        struct lu_fid   cr_fid1;
        struct lu_fid   cr_fid2;
        struct lustre_handle cr_old_handle; /* u64 handle in case of open replay */
        __u64           cr_time;
        __u64           cr_rdev;
```

```
        __u64           cr_ioepoch;
        __u64           cr_padding_1;
        __u32           cr_suppgid;
        __u32           cr_bias;
        __u32           cr_padding_2;
        __u32           cr_padding_3;
        __u32           cr_padding_4;
        __u32           cr_padding_5;
};

struct mdt_rec_link {
        __u32           lk_opcode;
        __u32           lk_fsuid;
        __u32           lk_fsgid;
        __u32           lk_cap;
        __u32           lk_suppgid1;
        __u32           lk_suppgid2;
        struct lu_fid   lk_fid1;
        struct lu_fid   lk_fid2;
        __u64           lk_time;
        __u64           lk_padding_1;
        __u64           lk_padding_2;
        __u64           lk_padding_3;
        __u64           lk_padding_4;
        __u32           lk_bias;
        __u32           lk_padding_5;
        __u32           lk_padding_6;
        __u32           lk_padding_7;
        __u32           lk_padding_8;
        __u32           lk_padding_9;
};

struct mdt_rec_unlink {
        __u32           ul_opcode;
        __u32           ul_fsuid;
        __u32           ul_fsgid;
        __u32           ul_cap;
        __u32           ul_suppgid;
        __u32           ul_mode;
        struct lu_fid   ul_fid1;
        struct lu_fid   ul_fid2;
        __u64           ul_time;
        __u64           ul_padding_1;
        __u64           ul_padding_2;
        __u64           ul_padding_3;
        __u64           ul_padding_4;
```

```
        __u32           ul_bias;
        __u32           ul_padding_5;
        __u32           ul_padding_6;
        __u32           ul_padding_7;
        __u32           ul_padding_8;
        __u32           ul_padding_9;
};

struct mdt_rec_rename {
        __u32           rn_opcode;
        __u32           rn_fsuid;
        __u32           rn_fsgid;
        __u32           rn_cap;
        __u32           rn_suppgid1;
        __u32           rn_suppgid2;
        struct lu_fid   rn_fid1;
        struct lu_fid   rn_fid2;
        __u64           rn_time;
        __u64           rn_padding_1;
        __u64           rn_padding_2;
        __u64           rn_padding_3;
        __u64           rn_padding_4;
        __u32           rn_mode;      /* cross-ref rename has mode */
        __u32           rn_bias;      /* some operation flags */
        __u32           rn_padding_5;
        __u32           rn_padding_6;
        __u32           rn_padding_7;
        __u32           rn_padding_8;
};
```

struct mdt_rec_setxattr is newly added:

```
 struct mdt_rec_setxattr {
        __u32           sx_opcode;
        __u32           sx_fsuid;
        __u32           sx_fsgid;
        __u32           sx_size;
        __u32           sx_flags;
        struct lu_fid   sx_fid;
        __u64           sx_padding_1;
        __u32           sx_padding_2;
        __u32           sx_padding_3;
        __u64           sx_valid;
        __u64           sx_padding_4;
        __u64           sx_padding_5;
```

```
        __u64           sx_padding_6;
        __u64           sx_padding_7;
        __u32           sx_padding_8;
        __u32           sx_padding_9;
        __u32           sx_padding_10;
        __u32           sx_padding_11;
        __u32           sx_padding_12;
        __u32           sx_padding_13;
};
```

5.3 64 bit alignment check

```
#define CHECK_VALUE_ALIGNMENT(a)                    \
do {                                                                    \
        printf("        LASSERTF("#a                                    \
                 " %% 8 == 0, \" got %%d\\n\",\n                "\
                 "%d %% 8);\n", a);                                     \
} while (0)

#define CHECK_MEMBER_ALIGNMENT(s,m)                             \
do {
        CHECK_VALUE_ALIGNMENT((int)offsetof(struct s, m));      \
} while(0)

#define CHECK_STRUCT_ALIGNMENT(s)                               \
do {
        CHECK_VALUE_ALIGNMENT((int)sizeof(struct s));           \
} while(0)
```

In CHECK_MEMBER(s, m), we will check alignment for each 64 bit field:

```
if (sizeof(((struct s *)0)->m) == 8)                                    \
                CHECK_MEMBER_ALIGNMENT(s, m);                   \
```

And in CHECK_STRUCT(s), we will do CHECK_STRUCT_ALIGNMENT(s) above.

5.4 struct ptlrpc_request

```
struct ptlrpc_request {
        ...
        /* request format */
        struct req_capsule    rq_pill;
        int                             rq_buf_nr;
```

```
        int                             rq_buf_size[REQ_MAX_FIELD_NR];
}
```

These fields are all request format related, will be used to pack/reserve reply.

5.5 int lustre_shrink_msg(struct lustre_msg *msg, int segment, unsigned int newlen, int move_data)

```
...
if (newlen == 0 && msg->lm_bufcount > segment + 1) {
        memmove(&msg->lm_buflens[segment], &msg->lm_buflens[segment + 1],
                        (msg->lm_bufcount - segment - 1) * sizeof(__u32));
        msg->lm_buflens[msg->lm_bufcount - 1] = 0;
}
```

The listed lines of code will be removed in 1.8 code, thereafter 1.8 server won't shrink lm_buflens.

5.6 void req_capsule_shrink(struct req_capsule *pill, const struct req_msg_field *field, unsigned int newlen, enum req_location loc)

```
LASSERT(req_capsule_has_field(pill, field, loc));
        offset = __req_capsule_offset(pill, field, loc);
        msg = __req_msg(pill, loc);
        len = max(field->rmf_size, 0);
        LASSERT(newlen <= len);
        loc == RCL_CLIENT ? pill->rc_req->rq_reqlen : pill->rc_req->rq_replen =
                lustre_shrink_msg(msg, offset, newlen, 1);
```

Currently we only do shrink on server side for the reply.

5.7 void mdt_shrink_reply(struct mdt_thread_info *info)

```
body = req_capsule_server_get(pill, &RMF_MDT_BODY);
if (req_capsule_has_field(pill, &RMF_MDT_MD, RCL_SERVER))
                 req_capsule_shrink(pill, &RMF_MDT_MD, body->eadatasize, RCL_SERVER);
if (req_capsule_has_field(pill, &RMF_ACL, RCL_SERVER))
                req_capsule_shrink(pill, &RMF_ACL, body->aclsize, RCL_SERVER);
if (req_capsule_has_field(pill, &RMF_CAPA1, RCL_SERVER) &&
    !(body->valid & OBD_MD_FLMDSCAPA))
                req_capsule_shrink(pill, &RMF_CAPA1, 0, RCL_SERVER);
```

```
if (req_capsule_has_field(pill, &RMF_CAPA2, RCL_SERVER) &&
    !(body->valid & OBD_MD_FLOSSCAPA))
                req_capsule_shrink(pill, &RMF_CAPA2, 0, RCL_SERVER);
```

This function will be called in mdt_getattr / mdt_getattr_name / mdt_reint / mdt_intent_getattr / mdt_reconstruct / mdt_close.

5.8 ptlrpc_request_alloc/free and ptlrpc_pack_request

```
struct ptlrpc_request *ptlrpc_request_alloc(struct obd_import *imp,
                                                                struct ptlrpc_request_p
{
        struct ptlrpc_request *request = NULL;
        if (pool)
                request = ptlrpc_prep_req_from_pool(pool);
        if (!request)
                OBD_ALLOC(request, sizeof(*request));
        if (request) {
                LASSERT((unsigned long)imp > 0x1000);
                LASSERT(imp != LP_POISON);
                LASSERT((unsigned long)imp->imp_client > 0x1000);
                LASSERT(imp->imp_client != LP_POISON);
                request->rq_import = class_import_get(imp);
        } else {
                CERROR("request allocation out of memory\n");
        }
        return request;
}
void ptlrpc_request_free(struct ptlrpc_request *request)
{
        if (request->rq_pool)
                __ptlrpc_free_req_to_pool(request);
        else
                OBD_FREE(request, sizeof(*request));
}
```

The above two functions do request allocation/free, besides them, a new function ***ptlrpc_pack_req***() will do other work in current ptlrpc_prep_req(), the code is the same.

5.9 int req_capsule_fill_sizes(struct req_capsule *pill, enum req_location *loc)

```
        const struct req_format *fmt = pill->rc_fmt;
```

```
        int                      i;
        LASSERT(fmt != NULL);
        for (i = 0; i < fmt->rf_fields[loc].nr; ++i) {
                if (pill->rc_area[i] == -1) {
                        pill->rc_area[i] = fmt->rf_fields[loc].d[i]->rmf_size;
                        if (pill->rc_area[i] == -1) {
                                LASSERT(loc != RCL_SERVER);
                                /* skip the following fields */
                                break;
                        }
                }
        }
        return i;
```

This function is to fill request.rq_pill.rc_area. It will be called in three functions: ***req_capsule_pack***() (which is renamed to ***req_capsule_server_pack***()), ***ptlrpc_pack_req***() and ***ptlrpc_req_set_repsize***(), eg:

```
static inline void ptlrpc_req_set_repsize(struct ptlrpc_request *req)
{
        int count = req_capsule_fill_sizes(&req->rq_pill, RCL_SERVER);
        req->rq_replen = lustre_msg_size(req->rq_reqmsg->lm_magic, count,
                                         req->rq_pill.rc_area);
        if (req->rq_reqmsg->lm_magic == LUSTRE_MSG_MAGIC_V2)
                req->rq_reqmsg->lm_repsize = req->rq_replen;
}
```

5.10 mdc_alloc/free/pack_req()

```
struct ptlrpc_request *mdc_alloc_req(struct obd_export *exp,
                                                        const struct req_format *format
{
        struct ptlrpc_request *req;
        req = ptlrpc_request_alloc(class_exp2cliimp(exp), NULL);
        if (req == NULL)
                return NULL;
        mdc_init_req_capsule(&req->pill, format);
        return req;
}
void mdc_free_req(struct ptlrpc_request *req)
{
        ptlrpc_request_free(req);
}
```

```
struct ptlrpc_request *mdc_pack_req(struct ptlrpc_request *req,
                                                        int version, int opc)
{
        struct req_capsule *pill = &req->rq_pill;
        int                 rc;
        rc = ptlrpc_pack_req(req, version, opc, NULL, NULL, NULL);
        if (rc)
                return rc;
        /* later this temp buf will be used to calculate size for reply */
        for (i = 0; i < ARRAY_SIZE(pill->rc_area); i++)
                pill->rc_area[i] = -1;
        return 0;
}
```

Through the above functions we could now use ***ptlrpc_request.rq_pill*** to prepare request buffers.

5.11 int mdc_getattr(struct obd_export *exp, const struct lu_fid *fid, struct obd_capa *oc, obd_valid valid, int ea_size, struct ptlrpc_request **request)

```
        struct ptlrpc_request *req;
        int                    rc;
        ENTRY;
        req = mdc_alloc_req(exp, &RQF_MDS_GETATTR);
        if (req == NULL)
                RETURN(-ENOMEM);
        mdc_set_capa_size(req, &RMF_CAPA1, oc);
        rc = mdc_pack_req(req, LUSTRE_MDS_VERSION, MDS_GETATTR);
        if (rc) {
                mdc_free_req(req);
                RETURN(rc);
        }
        /* MDS_BFLAG_EXT_FLAGS: request "new" flags(bug 9486) */
        mdc_pack_body(req, fid, oc, valid, ea_size, MDS_BFLAG_EXT_FLAGS);
        req_capsule_set_size(&req->rq_pill, &RMF_MDT_MD, RCL_SERVER, ea_size);
        if (valid & OBD_MD_FLRMTPERM)
                req_capsule_set_size(&req->rq_pill, &RMF_ACL, RCL_SERVER,
                                     sizeof(struct mdt_remote_perm));
        ptlrpc_req_set_repsize(req);
        rc = mdc_getattr_common(req);
        if (rc) {
                ptlrpc_req_finished(req);
                req = NULL;
        }
```

```
*request = req;
RETURN(rc);
```

With req_capsule all mdc code will prepare request in this way.

6 Focus for inspections

- anything missing?

7 ChangeLog

- 07-05-07 draft finished.
- 07-05-09 revision after nikita's inspection: fixed shrink mistakes; pack request by the pill itself instead of a temporary one in stack; add more description on request pack.
- 07-05-11 revision after nikita's inspection: rearrange the req_capsule size filling to avoid duplicate code, add use case for mdc_setattr.
- 07-05-14 revision after nikita's inspection: typo fixed in req_capsule_fill_sizes().