Main thread body for service threads. Waits in a loop waiting for new requests to process to appear. Every time an incoming requests is added to its queue, a waitq is woken up and one of the threads will handle it.

**static int ptlrpc\_main(void \*arg)**

**ptlrpc\_start\_thread(svcpt, 0);**

**ptlrpc\_server\_handle\_req\_in(svcpt, thread);**

**task = kthread\_run(ptlrpc\_main, thread, "%s", thread->t\_name);**

**static struct obd\_ops ost\_obd\_ops = {**

**.o\_owner = THIS\_MODULE,**

**.o\_setup = ost\_setup,**

**.o\_cleanup = ost\_cleanup,**

**.o\_health\_check = ost\_health\_check,**

**};**

**static int ost\_setup(struct obd\_device \*obd, struct lustre\_cfg\* lcfg)**

**struct ptlrpc\_service \***

**ptlrpc\_register\_service(struct ptlrpc\_service\_conf \*conf,**

**struct proc\_dir\_entry \*proc\_entry)**

**int ptlrpc\_start\_threads(struct ptlrpc\_service \*svc)**

**satic int ldlm\_set(void)**

**module\_init(ost\_init);**

**static int \_\_init ost\_init(void)**

**satic int mds\_start\_ptlrpc\_service(struct mds\_service \*m)**

**static int mgs\_init0(const struct lu\_env \*env, struct mgs\_device \*mgs, struct lu\_device\_type \*ldt, struct lustre\_cfg \*lcfg)**

**static int ptlrpc\_server\_handle\_request(**

**struct ptlrpc\_service\_part \*svcpt, struct ptlrpc\_thread \*thread)**

**以某种策略取得request，然后处理请求**

**static struct ptlrpc\_request \*ptlrpc\_server\_request\_get(struct ptlrpc\_service\_part \*svcpt, bool force)**

**static inline struct ptlrpc\_request \*ptlrpc\_nrs\_req\_get\_nolock(struct ptlrpc\_service\_part \*svcpt, bool hp, bool force)**

**{**

**return ptlrpc\_nrs\_req\_get\_nolock0(svcpt, hp, false, force);**

**}**

**static inline struct ptlrpc\_nrs\_request \* nrs\_request\_get**

**(struct ptlrpc\_nrs\_policy \*policy, bool peek, bool force)**

**.so\_req\_handler= tgt\_request\_handle**

**svc->srv\_ops.so\_req\_handler(request);**

**static int ptlrpc\_server\_request\_add(structptlrpc\_service\_part \*svcpt, struct ptlrpc\_request \*req)**

**参见NRS TBF阅读笔记中详细分析**

-

**static int tgt\_handle\_request0(struct tgt\_session\_info \*tsi,**

**struct tgt\_handler \*h, struct ptlrpc\_request \*req)**

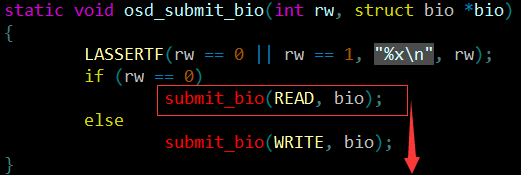
**rc = h->th\_act(tsi);**

**target\_committed\_to\_req(req);**

ofd\_commitrw lustre/ofd

Commit bulk IO to the storage.是ofd\_preprw()的伙伴函数. It finishes bulk IOrequest processing by committing buffers to the storage (WRITE) and/or freeing those buffers (read/write). See ofd\_commitrw\_read() and

ofd\_commitrw\_write() for details about each type of IO.



static struct tgt\_handler **ofd\_tgt\_handlers**[] = {

TGT\_OST\_HDL\_HP(HABEO\_CORPUS| HABEO\_REFERO, OST\_BRW\_READ, tgt\_brw\_read, ofd\_hp\_brw),

TGT\_OST\_HDL\_HP(HABEO\_CORPUS| MUTABOR, OST\_BRW\_WRITE, tgt\_brw\_write, ofd\_hp\_brw),

**经过这个宏ofd\_hp\_brw等最终会被赋值给tgt\_handler的int (\*th\_act)(struct tgt\_session\_info \*tsi);成员**

int tgt\_brw\_read(struct tgt\_session\_info \*tsi)

进入Linux内核函数调用

osd\_do\_bio

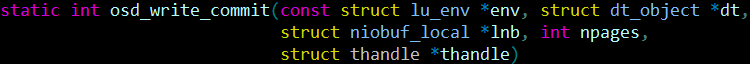
lustre/osd-ldiskfs

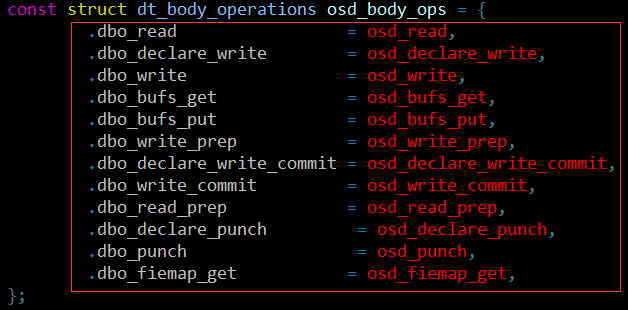


call

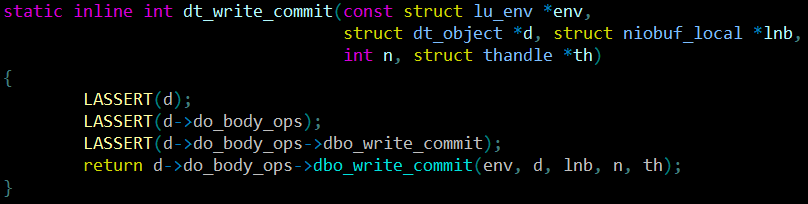
osd\_write\_commit

lustre/osd-ldiskfs





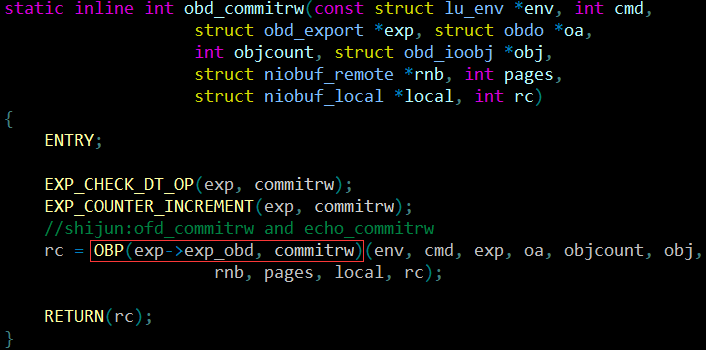
call



call

ofd\_commitrw\_read

ofd\_commitrw\_write



int tgt\_brw\_write(struct tgt\_session\_info \*tsi)

ptlrpc\_start\_bulk\_transfer

target\_bulk\_io

Lnet

两条线：一条提交到磁盘，一条开始传输（见下面分析）



ofd\_tgt\_handlers[]

**h获得：也即选择哪种处理**

static int ptlrpc\_server\_handle\_request(struct ptlrpc\_service\_part \*svcpt, struct ptlrpc\_thread \*thread)

{

…svc->srv\_ops.so\_req\_handler(request);…

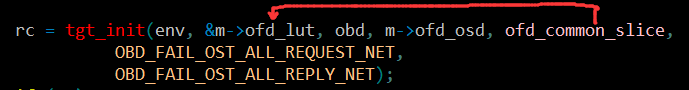
}

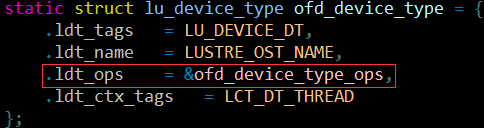
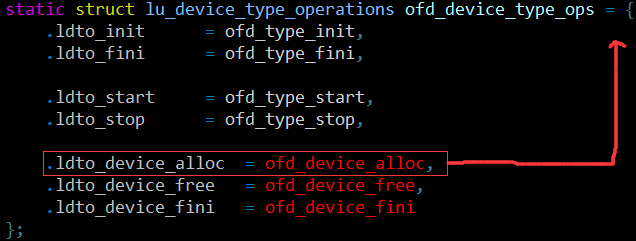
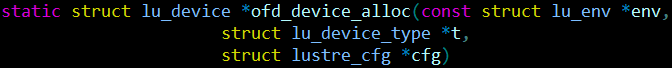
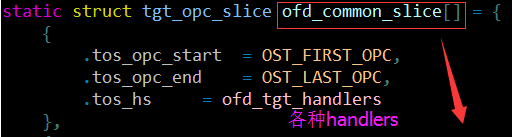
**int tgt\_request\_handle(struct ptlrpc\_request \*req)**

**static int ofd\_init0(const struct lu\_env \*env, struct ofd\_device \*m,**

**struct lu\_device\_type \*ldt, struct lustre\_cfg \*cfg)**

lustre/ofd/ofd\_dev.c





**module\_init(ofd\_init);**

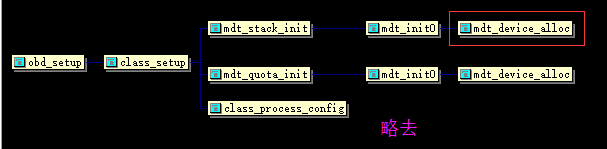
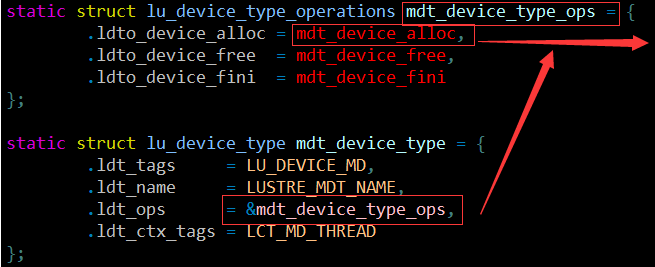


type->typ\_lu = ldt;

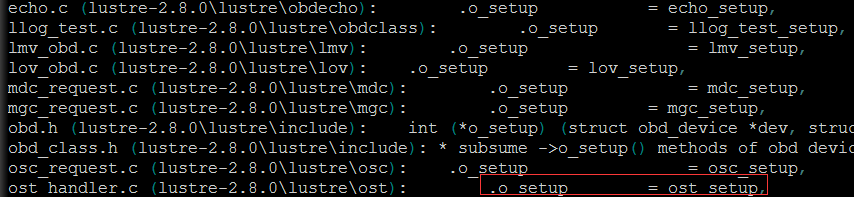
rc = lu\_device\_type\_init(ldt);

static int \_\_init mdt\_init(void)

module\_init(mdt\_init);



d = ldt->ldt\_ops->**ldto\_device\_alloc**(&env, ldt, cfg);



**.so\_thr\_init= tgt\_io\_thread\_init,**

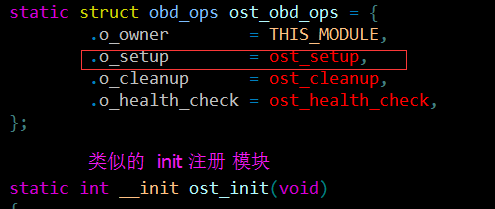
**.so\_thr\_done = tgt\_io\_thread\_done,**

**.so\_req\_handler = tgt\_request\_handle,**

**.so\_hpreq\_handler = tgt\_hpreq\_handler,**

**.so\_req\_printer = target\_print\_req,**

**等等 类似的操作**



**static int tgt\_handle\_request0(struct tgt\_session\_info \*tsi,**

**struct tgt\_handler \*h, struct ptlrpc\_request \*req)**

**rc = h->th\_act(tsi);**

**target\_committed\_to\_req(req);**

**h** = tgt\_handler\_find\_check(req); 获得h

ofd\_init0

rc = OBP(obd, setup)(obd, cfg);

**struct file\_operations ll\_file\_operations\_noflock = {**

**//或者是ll\_file\_operations\_flock**

**... ...**

**.read = ll\_file\_read,**

**.aio\_read = ll\_file\_aio\_read,**

**.write = ll\_file\_write,**

**.aio\_write = ll\_file\_aio\_write,**

**... ...};**

之前的操作参见lustre文件系统注册操作

以**ll\_file\_write**为例

int cl\_io\_loop(const struct lu\_env \*env, struct cl\_io \*io)

int ptlrpcd\_queue\_work(void \*handler));

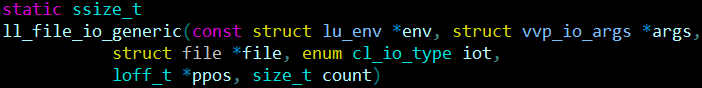
static void osc\_check\_rpcs(const struct lu\_env \*env, struct client\_obd \*cli)

int osc\_extent\_release(const struct lu\_env \*env, struct osc\_extent \*ext)

**ll\_cl\_add(file, env, io);**

**rc = cl\_io\_loop(env, io);**

**ll\_cl\_remove(file, env);**



scan->cis\_iop->op[io->ci\_type].cio\_end(env, scan);

通过结构体赋值方式

static const struct cl\_io\_operations osc\_io\_ops = {

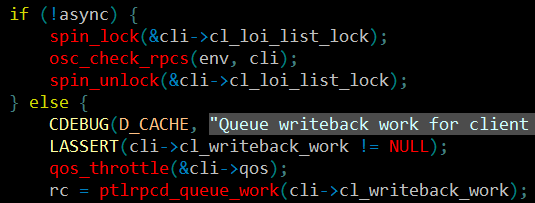
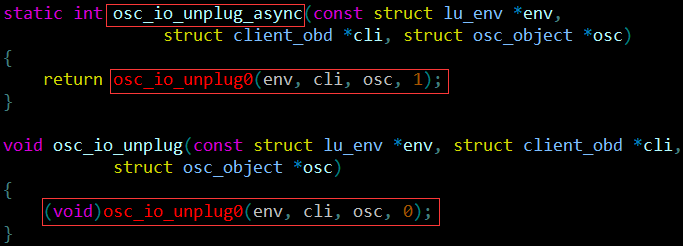
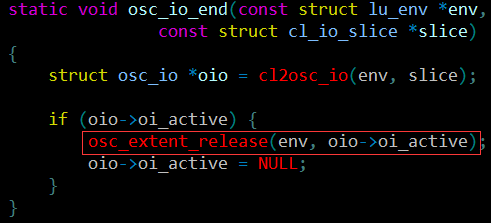
**. . . . . .**

.cio\_start = osc\_io\_write\_start,

.cio\_end = osc\_io\_end,

**. . . . . .**

void cl\_io\_end(const struct lu\_env \*env, struct cl\_io \*io)



void ptlrpc\_set\_add\_new\_req**(**struct ptlrpcd\_ctl **\***pc**,**

struct ptlrpc\_request **\***req**)**

**{**

**list\_add\_tail(&req->rq\_set\_chain,&set->set\_new\_requests);**

**if** **(**count **==** 1**)** **{**

**wake\_up(&set->set\_waitq);**

**for** **(**i **=** 0**;** i **<** pc**->**pc\_npartners**;** i**++)**

**wake\_up(&pc->pc\_partners[i]->pc\_set->set\_waitq);**

**}**

**}**

void ptlrpcd\_add\_req(struct ptlrpc\_request \*req)

req->rq\_commit\_cb = brw\_commit;

req->rq\_interpret\_reply = brw\_interpret;

qos\_throttle(&cli->qos);

**ptlrpcd\_add\_req(req);**

static int osc\_send\_write\_rpc(const struct lu\_env \*env,

struct client\_obd \*cli,struct osc\_object \*osc)

static int osc\_send\_read\_rpc(const struct lu\_env \*env,

struct client\_obd \*cli,struct osc\_object \*osc)

pc = ptlrpcd\_select\_pc(req);

**ptlrpc\_set\_add\_new\_req(pc, req);**

Requests that are added to the ptlrpcd queue are sent via ptlrpcd\_check->ptlrpc\_check\_set().

再往上参见前面的分析

static int ptlrpcd\_init(void)

int ptlrpcd\_start**(**struct ptlrpcd\_ctl **\***pc**)**

**{. . . . . .**

task **=** kthread\_run**(**ptlrpcd**,** pc**,**pc**->**pc\_name**);**

**. . . . . .}**

static int ptlrpcd\_check**(**struct lu\_env **\***env**,** struct ptlrpcd\_ctl **\***pc**)**

**{**

**......**

list\_splice\_init**(&**set**->**set\_new\_requests**,**

**&**set**->**set\_requests**);**//添加 链表链接挂载

**......**

list\_for\_each\_safe**(**pos**,** tmp**,** **&**set**->**set\_requests**)** **{**

req **=** list\_entry**(**pos**,** struct ptlrpc\_request**,** rq\_set\_chain**);**

**if** **(**req**->**rq\_phase **!=** RQ\_PHASE\_COMPLETE**)**

**break;**

list\_del\_init**(&**req**->**rq\_set\_chain**);**

req**->**rq\_set **=** **NULL;**

ptlrpc\_req\_finished**(**req**);**

**}......**

**}**

static int ptlrpcd(void \*arg){

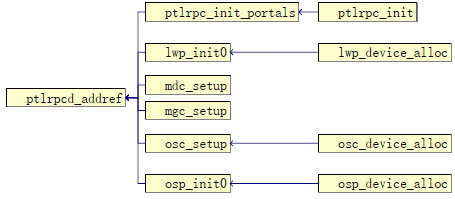
struct ptlrpcd\_ctl \*pc = arg;

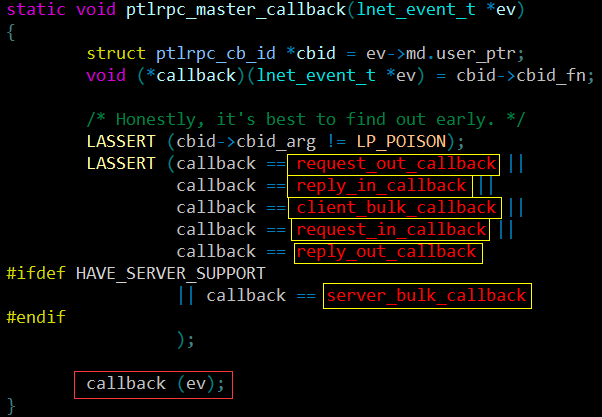
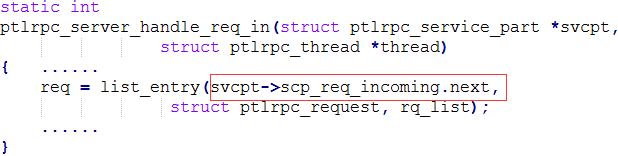
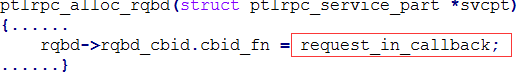
**. . . . . .**

l\_wait\_event(set->set\_waitq,**ptlrpcd\_check(&env, pc)**, &lwi);

**. . . . . .**

}





后续操作参见相关分析

类似的操作

rc = LNetEQAlloc(0, **ptlrpc\_master\_callback**, &ptlrpc\_eq\_h)

ptlrpc\_master\_callback

ptlrpc\_ni\_init

ptlrpc\_init\_portals

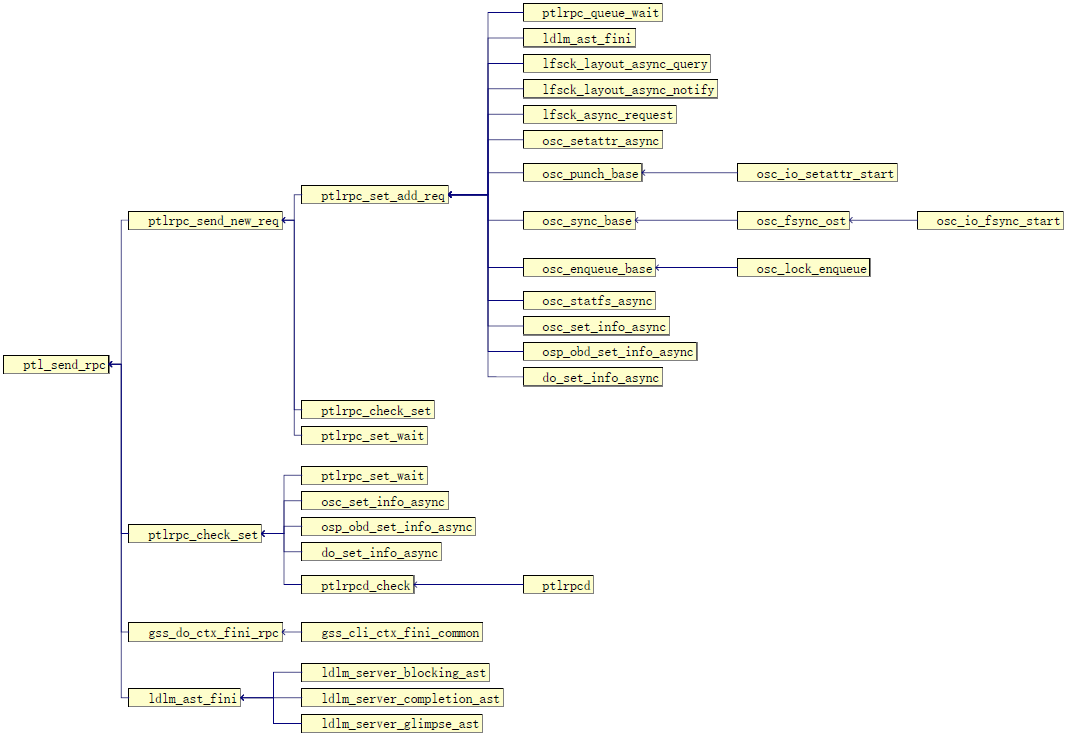
**struct ptlrpc\_request**

**struct list\_head set\_requests**

**struct list\_head cr\_set\_chain**

**struct ptlrpc\_request\_set \*cr\_set**

**struct ptlrpc\_cli\_req rq\_cli**



Lnet

ptlrpc\_master\_callback

ptlrpc\_master\_callback

ptlrpc\_master\_callback