drivers/mtd/mtdblock.c

## mtdblock

struct mtd\_blktrans\_ops mtdblock\_tr = {

.name = “mtdblock”,

.major = MTD\_BLOCK\_MAJOR,

.add\_mtd = mtdblock\_add\_mtd,

.part\_bits =0,

.flush = mtdblock\_flush,

.readsect = mtdblock\_readsect,

.writesect = mtdblock\_writesect,

};

struct mtd\_notifier blktrans\_notifier = {

.add = blktrans\_notify\_add,

.remove = blktrans\_nofity\_remove,

};

init\_mtdblock->

register\_mtd\_blktrans(&mtdblock\_tr)

{

struct mtd\_blktrans\_ops \*ts = &mtdblock\_tr;

(if!blktrans\_notifier.list.next)

register\_mtd\_user(&blktrans\_notifier);

list\_add(&tr,&blktrans\_majors)

}

register\_mtd\_user(&blktrans\_notifier)

{

struct mtd\_notifier \*new = &blktrans\_notifier;

list\_add(&new->list,&mtd\_notifier);

mtd\_for\_each\_device(mtd)

mtd->add(mtd);

}

## mtd\_add\_device\_partitions

drivers/mtd/mtdcore.c

mtd\_add\_device\_partitions ->

add\_mtd\_partition(mtd,real\_parts,nbparts)

{

struct mtd\_info \*master = mtd;

struct mtd\_partition \*parts = real\_parts;

struct mtd\_part \*slave;

int i;

uint64\_t cur\_offset =0;

for(i=0;i<nbparts;i++) {

slave = allocate\_partition(master,parts+i,cur\_offset);

list\_add(&slave->list,&mtd\_partitions);

add\_mtd\_device(&slave->mtd);

cur\_offset = slave->offset + slave->mtd.size;

}

}

struct backing\_dev\_info mtd\_bdi = {

};

add\_mtd\_device(&slave->mtd)

{

struct mtd\_info \*mtd = &slave->mtd;

int i;

struct mtd\_notifier \*not;

mtd->backing\_dev\_info = &mtd\_bdi;

i = idr\_alloc(&mtd\_idr,mtd,0,0,GFP\_KERNEL);

list\_for\_each\_entry(not,&mtd\_notifier,list)

not->add(mtd);

}

not->add(mtd)->

blktrans\_notify\_add(mtd)

{

struct mtd\_blktrans\_ops \*tr;

list\_for\_each\_entry(tr,\*blktrans\_majors,list)

tr->add\_mtd(tr,mtd);

}

struct mtdblk\_dev {

struct mtd\_blktrans\_dev mbd;

}

tr->add\_mtd(tr,mtd)->

mtdblock\_add\_mtd(tr,mtd)

{

struct mtdblk\_dev \*dev = kzalloc(sizeof(\*dev),GFP\_KERNEL);

add\_mtd\_blktrans\_dev(&dev->mbd);

}

struct block\_device\_operations mtd\_block\_ops = {

.owner = THIS\_MODULE,

.open = blktrans\_open,

.getgeo = blktrans\_getgeo,

};

add\_mtd\_blktrans\_dev(&dev->mbd)

{

struct mtd\_blktrans\_dev \*new = &dev->mbd;

struct gendisk \*gd;

gd = alloc\_disc(1<<tr->part\_bits);

gd->private\_data = new;

gd->major = tr->major;

gd->first\_minor = (new->devnum) << tr->part\_bits;

gd->fops = &mtd\_block\_ops;

snprintf(gd->disk\_name,sizeof(gd->disk\_name),”%s%d”,tr->name,new->devnum);

new->rq = blk\_init\_queue(mtd\_blktrans\_request,&new->queue\_lock)

if(tr->flush)

blk\_queue\_write\_cache(new->rq,true,false);

new->rq->queuedata = new;

if(tr->discard) {

}

new->wq =alloc\_workqueue(“%s%d”,0,0,tr->name,new->mtd->index);

INIT\_WORK(&new->work,mtd\_blktrans\_work);

device\_add\_disk(&new->mtd->dev,gd);

}