

differential block device Reference Manual
(tchize@mailandnews.com)

Generated by Doxygen 1.2.9.1

Tue Jun 4 13:41:49 2002

Contents

1	differential block device Data Structure Index	1
2	differential block device File Index	1
3	differential block device Page Index	2
4	differential block device Data Structure Documentation	2
5	differential block device File Documentation	10
6	differential block device Page Documentation	50

1 differential block device Data Structure Index

1.1 differential block device Data Structures

Here are the data structures with brief descriptions:

diffdev_device (Virtual device datas)	2
diffdev_entry (Diffdev physical device specific datas)	3
diffdev_params (Ioctl params stucture)	6
diffdev_superblock_entry (Shield super block)	6
diffdev_table_entry (The following structure is used to define a 'shield' for a sector)	7
sub_device_op (Used to keep track of a sub operation)	7
superblock_op (Used to delay a sub operation)	8
superblock_w_op (Super block saving structure)	9

2 differential block device File Index

2.1 differential block device File List

Here is a list of all documented files with brief descriptions:

diffmod.c (Main device file)	10
diffmod.h (Structures and macros definition)	26

<code>difftransfert.c</code> (Transfert toolkit)	31
<code>diffutil.c</code> (Device toolkit)	41

3 differential block device Page Index

3.1 differential block device Related Pages

Here is a list of all related documentation pages:

Todo List	50
-----------	----

4 differential block device Data Structure Documentation

4.1 `diffdev_device` Struct Reference

Virtual device datas.

```
#include <diffmod.h>
```

Data Fields

- unsigned long **flags**
various flags.
- block_device * **bdev_shadow**
block device being proteted.
- kdev_t **kdev_shadow**
k_dev.t of block device being proteted.
- block_device * **bdev_shield**
block device serving as shield.
- kdev_t **kdev_shield**
k_dev.t of block device serving as shield.
- `diffdev_entry` * **head**
Physical datas layout.
- int **open**
Knows if device is already opened.
- int **valid**

Know if datas in this structure are valids.

- semaphore **sem_config**
Configuration semaphore used for concurrent ioctl prevention.
- rwlock_t **rwlock_config**
Configuration read/write spinlock.
- char * **shieldname**
Name of device we are protecting.
- char * **shadowname**
Name of device servind as shield.

4.1.1 Detailed Description

Virtual device datas.

This structure contain informations about a virtual diffdev device, a pointer to physical data, a spinlock for SMP protection. and a semaphore for concurrent ioctl prevention.

Definition at line 251 of file diffmod.h.

4.1.2 Field Documentation

4.1.2.1 char* shieldname

Name of device we are protecting.

ex: /dev/hda1

Definition at line 274 of file diffmod.h.

The documentation for this struct was generated from the following file:

- **diffmod.h**

4.2 diffdev_entry Struct Reference

diffdev physical device specific datas.

```
#include <diffmod.h>
```

Data Fields

- int **magic**
Non disk saved various datas Magic number as recommended for developing structure.

- **spinlock_t lock**
structure lock.
- unsigned long **nr_block_shield**
Number of blocks in shield.
- unsigned long **nr_block_shadow**
Number of blocks in shadow.
- unsigned long **hardsect_size**
size of each sector.
- unsigned long **blksize**
size of each block.
- **diffdev_superblock_entry * superblock**
For each superblock, list of blocks.
- unsigned long * **superflags**
flags for various action related to superblocs.
- **__u32 physical_address**
Physical address where table is saved.
- **__u32 header_size**
Size used for this virtual device header.
- **__u32 block_span**
Number of block the physical datas spans over.
- void * **physical_datas**
Pointer to first byte in this struct that has to be saved.
- int **needsave**
When set to 1, table must be saved cause altered.
- **__u32 * superblock_addr**
For each superblock, it's physical address where saved, 0 means doesn't exist yet.
- char **signature** [7]
*Disk saved data. Continuous so may be mapped directly with disk (Yeah!)
signature used to identify a diffdev block.*
- char **version** [7]

Version used to create the diffdev.

- **char id [32]**
name given to transaction / savepoint.
- **__u32 superblock_size**
Quantity of blocks saved in a superblock.
- **__u32 current_free_block**
number of first free block.
- **__u32 next_table**
Physical address of next table(savepoint).
- **__u32 superblock_count**
Number of superblocks in the table.

4.2.1 Detailed Description

diffdev physical device specific datas.

Some of the datas are used only in memory. Others have to be saved to disk. Those are grouped at the end of the structure to be mapped directly on the disk.

Definition at line 186 of file diffmod.h.

4.2.2 Field Documentation

4.2.2.1 int magic

Non disk saved various datas Magic number as recommended for developping structure.

See linux doc: "magicnumber.txt". should be MAGIC_DIFFDEV_ENTRY

Definition at line 193 of file diffmod.h.

4.2.2.2 unsigned long* superflags

flags for various action related to superblocs.

(e.g. dirty flag, loading flag, awaiting flag)

Definition at line 207 of file diffmod.h.

4.2.2.3 __u32* superblock_addr

For each superblock, it's physical address where saved, 0 means doesn't exist yet.

superblock_addr always points at end of its diffdev_entry

Definition at line 222 of file diffmod.h.

4.2.2.4 __u32 next_table

Physical address of next table(savepoint).

May be null if current is last

Definition at line 237 of file diffmod.h.

The documentation for this struct was generated from the following file:

- **diffmod.h**

4.3 diffdev_params Struct Reference

Ioctl params stucture.

```
#include <diffmod.h>
```

Data Fields

- int **device_nbr**
- char **shadowname** [MAX_DEVICE_NAME_LEN]
- char **shieldname** [MAX_DEVICE_NAME_LEN]

4.3.1 Detailed Description

Ioctl params stucture.

Structure used to get or send configuration about a device using the ioctl DIFFDEV_IO_GETPARAMS and DIFFDEV_IO_SETPARAMS commands

Definition at line 40 of file diffmod.h.

The documentation for this struct was generated from the following file:

- **diffmod.h**

4.4 diffdev_superblock_entry Struct Reference

shield super block.

```
#include <diffmod.h>
```

Data Fields

- **diffdev_table_entry * block**

4.4.1 Detailed Description

shield super block.

'shield' are grouped into a superblock entry. See **diffdev_table_entry** (p. 7). Perhaps i should have put the superflags here... But this could lead to problems concerning flushing super blocks...

Definition at line 141 of file diffmod.h.

The documentation for this struct was generated from the following file:

- **diffmod.h**

4.5 diffdev_table_entry Struct Reference

The following structure is used to define a 'shield' for a sector.

```
#include <diffmod.h>
```

Data Fields

- int **MOD**:1
- int **COR**:1
- `__u32` **shield**:30

4.5.1 Detailed Description

The following structure is used to define a 'shield' for a sector.

Note that no magic is present!

Definition at line 129 of file diffmod.h.

The documentation for this struct was generated from the following file:

- **diffmod.h**

4.6 sub_device_op Struct Reference

Used to keep track of a sub operation.

```
#include <diffmod.h>
```

Data Fields

- int **magic**
magic number.
- `buffer_head` * **request**

request call that leads to performing a subdevice op.

- **diffdev_device * device**

device concerned by operation.

- **int op_type**

type of operation (READ, READA or WRITE).

- **int try_count**

when zero, stop trying.

4.6.1 Detailed Description

Used to keep track of a sub operation.

When a sub operation is done, this structure contain the information about the apparent operation. This is used to terminate the transfert.

Definition at line 336 of file diffmod.h.

4.6.2 Field Documentation

4.6.2.1 int magic

magic number.

Should be MAGIC_DIFFDEV_SUBOP

Definition at line 340 of file diffmod.h.

The documentation for this struct was generated from the following file:

- **diffmod.h**

4.7 superblock_op Struct Reference

Used to delay a sub operation.

```
#include <diffmod.h>
```

Data Fields

- **int magic**

magic number.

- **buffer_head * request**

requested call that lead to read a superblock.

- **diffdev_device * device**

device concerned.

- int **try_count**
when zero stops trying to load.
- int **superblock_nr**
superblock concerned.
- int **op_type**
type of operation (READ, READA or WRITE) the request is.
- superblock_op * **next**
next awaiting order.

4.7.1 Detailed Description

Used to delay a sub operation.

This structure is read when a superblock has finished loading. this is a linked list of awaiting orders!

Definition at line 312 of file diffmod.h.

4.7.2 Field Documentation

4.7.2.1 int magic

magic number.

Should be MAGIC_DIFFDEV_SBLKOP

Definition at line 316 of file diffmod.h.

The documentation for this struct was generated from the following file:

- **diffmod.h**

4.8 superblock_w_op Struct Reference

super block saving structure.

```
#include <diffmod.h>
```

Data Fields

- int **magic**
magic number.
- diffdev_device * **device**

device we did the operation for.

- `__u32 superbblock_nr`
superblock number we did save.

4.8.1 Detailed Description

super block saving structure.

This is used to know which device we saved the superblock from. Used to check the needsave flag and clear the saving flag.

Definition at line 298 of file diffmod.h.

4.8.2 Field Documentation

4.8.2.1 int magic

magic number.

Should be MAGIC_DIFFDEV_SBLKW

Definition at line 302 of file diffmod.h.

The documentation for this struct was generated from the following file:

- `diffmod.h`

5 differential block device File Documentation

5.1 diffmod.c File Reference

Main device file.

```
#include <linux/autoconf.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/blk.h>
#include <linux/blkdev.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/spinlock.h>
#include <linux/ioctl.h>
#include <linux/string.h>
```

```
#include <linux/devfs_fs_kernel.h>
#include <linux/fs.h>
#include <asm/io.h>
#include <asm/processor.h>
#include <asm/uaccess.h>
#include <linux/version.h>
#include "diffmod.h"
#include "diffproto.h"
```

Defines

- #define **MODULE**
Needed to create a module.
- #define **MAJOR_NR** `diffdev_major`
Macro used by blk.h to get the major number.
- #define **DEVICE_NAME** `"diffdev"`
Name of the device we will reference.
- #define **DEVICE_NR**(device) `MINOR(device)`
Used to get the subdevice associated with a file.
- #define **DEVICE_INTR** `diffdev_intrptr`
Can somebody tell me the use of this?
- #define **DEVICE_NO_RANDOM**
We do not provide chaos to the random pool.
- #define **DEVICE_REQUEST** `diffdev_request`
request function associated with this driver.
- #define **DEBUG**
- #define **ACQUIRE_CONFIG**(__devnbr) `down_interruptible`
`(&(diffdevs[__devnbr].sem_config))`
Acquire the config semaphore for a specific device.
- #define **RELEASE_CONFIG**(__devnbr) `up(&(diffdevs[__devnbr].sem_config))`
Free the config semaphore for a specific device.
- #define **CONFIG_READ_LOCK**(__devnbr, __flags) `read_lock_-`
`irqsave(&(diffdevs[__devnbr].rwlock_config),__flags)`
Acquire the config read spinlock for a specific device.

- #define **CONFIG_READ_UNLOCK**(__devnbr, __flags) read_unlock_
irqrestore(&(diffdevs[__devnbr].rwlock_config),__flags)
Free the config read spinlock for a specific device.
- #define **CONFIG_WRITE_LOCK**(__devnbr, __flags) write_lock_
irqsave(&(diffdevs[__devnbr].rwlock_config),__flags)
Acquire the config write spinlock for a specific device.
- #define **CONFIG_WRITE_UNLOCK**(__devnbr, __flags) write_
unlock_irqrestore(&(diffdevs[__devnbr].rwlock_config),__flags)
Free the config write spinlock for a specific device.
- #define **is_config_device**(_devnbr) (_devnbr==0)
- #define **check_diffdev_number**(_devnbr) ((_devnbr>0) && (-
devnbr<diffdev_devs))
- #define **diffdev_is_open**(_xyz) (check_diffdev_number(_xyz) &&
(diffdevs[_xyz].open))

Functions

- **MODULE_LICENSE** ("GPL")
This drivers is placed under GPL license.
- **MODULE_AUTHOR** ("tchize:tchize @mailandnews.com")
Authoring informations.
- **MODULE_DESCRIPTION** ("differential block device")
Short description of driver.
- **MODULE_PARM** (major_shadow,"i")
- **MODULE_PARM_DESC** (major_shadow,"major number of block de-
vice to protect")
- **MODULE_PARM_DESC** (minor_shadow,"minor number of block de-
vice to protect")
- **MODULE_PARM_DESC** (major_shield,"major number of block de-
vice to write to")
- **MODULE_PARM_DESC** (minor_shield,"minor number of block de-
vice to write to")
- int **diffdev_make_request** (request_queue_t *queue, int rw, struct
buffer_head *bh)
Generic make request function.
- int **diffdev_open** (struct inode *inode, struct file *filp)
Block device open function.
- int **diffdev_release** (struct inode *inode, struct file *filp)

Block device close function.

- int **valid_parameters** (**diffdev_params** *parms)
- int **diffdev_ioctl_params** (unsigned command, unsigned long argument)

Mess with device configuration ;-) This function sets or get the device configuration.

- int **diffdev_ioctl** (struct inode *inode, struct file *filp, unsigned command, unsigned long argument)

Ioctl entry point.

- int **diffdev_checkchange** (kdev_t dev)

Check media change.

- int **diffdev_revalidate** (kdev_t dev)

Revalidate.

- int **init_module** (void)

Initialisation stuff.

- void **cleanup_module** (void)

Module exit and cleanup point.

Variables

- int **diffdev_major**

Will save the major number of the block device.

- int **major_shadow** = -1
- int **minor_shadow** = -1
- int **major_shield** = -1
- int **minor_shield** = -1
- spinlock_t **diffdev_lock**

spinlock used in SMP to prevent concurrent access to device structures.

- int * **diffdev_blksizes** = NULL
- int * **diffdev_sizes** = NULL
- int * **diffdev_hardsects** = NULL
- int **diffdev_devs**
- int **diffdev_rahead**
- int **diffdev_size**
- int **diffdev_blksize**
- int **diffdev_hardsect**
- block_device * **shadow** = NULL

- `block_device * shield = NULL`
- `diffdev_device * diffdevs = NULL`
`diffdev_device` (p. 2) *array*.
- `atomic_t atome_config`
- `block_device_operations diffdev_bdops`
Block device operations.

5.1.1 Detailed Description

Main device file.

This file contain all functions needed to handle block device level operations. Transaction level operation will surely be put somewhere else...

Definition in file `diffmod.c`.

5.1.2 Define Documentation

5.1.2.1 `#define ACQUIRE_CONFIG(__devnbr) down_interruptible(&(diffdevs[__devnbr].sem_config))`

Acquire the config semaphore for a specific device.

Return true if interrupted.

Definition at line 108 of file `diffmod.c`.

Referenced by `diffdev_ioctl_params()`, `diffdev_open()`, and `diffdev_release()`.

5.1.2.2 `#define CONFIG_READ_LOCK(__devnbr, __flags) read_lock_irqsave(&(diffdevs[__devnbr].rwlock_config), __flags)`

Acquire the config read spinlock for a specific device.

Save the irq and prevent interruption

Definition at line 112 of file `diffmod.c`.

Referenced by `diffdev_make_request()`.

5.1.2.3 `#define CONFIG_READ_UNLOCK(__devnbr, __flags) read_unlock_irqrestore(&(diffdevs[__devnbr].rwlock_config), __flags)`

Free the config read spinlock for a specific device.

Restore the irq.

Definition at line 114 of file `diffmod.c`.

Referenced by `diffdev_make_request()`.

5.1.2.4 #define CONFIG_WRITE_LOCK(__devnbr, __flags) write_lock_irqsave(&(diffdevs[__devnbr].rwlock_config),__flags)

Acquire the config write spinlock for a specific device.

Save the irq and prevent interruption

Definition at line 116 of file diffmod.c.

Referenced by diffdev_ioctl_params(), diffdev_open(), and diffdev_release().

5.1.2.5 #define CONFIG_WRITE_UNLOCK(__devnbr, __flags) write_unlock_irqrestore(&(diffdevs[__devnbr].rwlock_config),__flags)

Free the config write spinlock for a specific device.

Restore the irq.

Definition at line 118 of file diffmod.c.

Referenced by diffdev_ioctl_params(), diffdev_open(), and diffdev_release().

5.1.3 Function Documentation

5.1.3.1 int diffdev_make_request (request_queue_t * queue, int rw, struct buffer_head * bh)

Generic make request function.

diffdev_make_request is the basic function called by block device system to execute block operations on this device.

Operations can be

- READ read blocks from device
- READA read ahead blocks from device
- WRITE write block to device

Definition at line 146 of file diffmod.c.

```

00147 {
00148     printk ("<1> Congratulation man, the kernel managed to use you own function :))\n");
00149     unsigned long irqflags;
00150     #ifdef CONFIG_HIGHMEM
00151     bh=create_bounce(rw,bh);
00152     #endif
00153     printk ("<1> request %p: cmd %i size %i (b_rsector. %li) next:%p\n",
00154           bh,
00155           rw,
00156           bh->b_size,
00157           bh->b_rsector,
00158           bh->b_reqnext);
00159     spin_lock_irqsave(&diffdev_lock,irqflags);
00160     if (!check_diffdev_number(DEVICE_NR(bh->b_rdev))){
00161         spin_unlock_irqrestore(&diffdev_lock,irqflags);
00162         bh->b_end_io(bh,0);
00163         printk ("<1> [diffdev] called diffdev_make_request with an invalid request number\n");

```

```

00164     DIFFDEV_MAILTO;
00165     return 0;
00166 }
00167 spin_unlock_irqrestore(&diffdev_lock,irqflags);
00168
00169 switch(rw){
00170     case READ:
00171     case READA:
00172         printk("<1>READ(A) called\n");
00173         CONFIG_READ_LOCK(DEVICE_NR(bh->b_rdev),irqflags);
00174         if (!diffdevs[DEVICE_NR(bh->b_rdev)].open){
00175             printk("<1> arg device %i is not opened\n",DEVICE_NR(bh->b_rdev));
00176             CONFIG_READ_UNLOCK(DEVICE_NR(bh->b_rdev),irqflags);
00177             bh->b_end_io(bh,0);
00178             break;
00179         }
00180         CONFIG_READ_UNLOCK(DEVICE_NR(bh->b_rdev),irqflags);
00181         //bh->b_end_io(bh,0);
00182         //return 0;
00183         try_read_block (bh,&(diffdevs[DEVICE_NR(bh->b_rdev)]));
00184         break;
00185     case WRITE:
00186         printk("<1>WRITE called\n");
00187         CONFIG_READ_LOCK(DEVICE_NR(bh->b_rdev),irqflags);
00188         if (!diffdevs[DEVICE_NR(bh->b_rdev)].open){
00189             printk("<1> arg device %i is not opened\n",DEVICE_NR(bh->b_rdev));
00190             CONFIG_READ_UNLOCK(DEVICE_NR(bh->b_rdev),irqflags);
00191             bh->b_end_io(bh,0);
00192             break;
00193         }
00194         CONFIG_READ_UNLOCK(DEVICE_NR(bh->b_rdev),irqflags);
00195         //bh->b_end_io(bh,0);
00196         //return 0;
00197         try_write_block (bh,&(diffdevs[DEVICE_NR(bh->b_rdev)]));
00198         break;
00199     default:
00200         printk("<1> :(( unknown operation required: %i\n",rw);
00201         bh->b_end_io(bh,0);
00202         break;
00203 }
00204 return 0; /*done :)*
00205 }

```

5.1.3.2 int diffdev_open (struct inode * *inode*, struct file * *filp*)

Block device open function.

Check the validity of the device program try to open. The main thing to test is concurrent use and validity of config parameters. We also increment module usage count here. Acquire:

- sem_config semaphore
- rwlock_config rw_lock for write in irqsave way

Definition at line 215 of file diffmod.c.

```
00216 {
```

```

00217 unsigned long flags;
00218 int result;
00219 printk (<1>diffdev_open called\n");
00220
00221 MOD_INC_USE_COUNT;
00222 spin_lock_irqsave(&diffdev_lock,flags);
00223 /* configuration device is always openable :) */
00224 if (is_config_device(DEVICE_NR(inode->i_rdev))
00225 {
00226     printk (<1>config device\n");
00227     atomic_inc (&atome_config);
00228     spin_unlock_irqrestore(&diffdev_lock,flags);
00229     return 0;
00230 }
00231 /*Valid device?*/
00232 if (!check_diffdev_number(DEVICE_NR(inode->i_rdev)){
00233     spin_unlock_irqrestore(&diffdev_lock,flags);
00234     MOD_DEC_USE_COUNT;
00235     return -ENODEV;
00236 }
00237 spin_unlock_irqrestore(&diffdev_lock,flags);
00238
00239
00240 /*Lock config*/
00241 if (ACQUIRE_CONFIG(DEVICE_NR(inode->i_rdev)){
00242     MOD_DEC_USE_COUNT;
00243     return -ERESTARTSYS;
00244 }
00245 CONFIG_WRITE_LOCK(DEVICE_NR(inode->i_rdev),flags);
00246 /*Device as been ioctl configured?*/
00247 if (!diffdevs[DEVICE_NR(inode->i_rdev)].valid){
00248     CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00249     RELEASE_CONFIG(DEVICE_NR(inode->i_rdev));
00250     MOD_DEC_USE_COUNT;
00251     return -EINVAL;
00252 }
00253 /*Device is not yet opened?*/
00254 if ( diffdev_is_open(DEVICE_NR(inode->i_rdev)){
00255     CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00256     RELEASE_CONFIG(DEVICE_NR(inode->i_rdev));
00257     MOD_DEC_USE_COUNT;
00258     return -EBUSY;
00259 }
00260 /*Mark device opened*/
00261 diffdevs[DEVICE_NR(inode->i_rdev)].open=1;
00262 CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00263 RELEASE_CONFIG(DEVICE_NR(inode->i_rdev));
00264 /*Check validity of sub devices*/
00265 printk (<1>Opening device shadow (%s)\n",diffdevs[DEVICE_NR(inode->i_rdev)].shadowname);
00266 if ((result=diffdev_try_open (
00267     diffdevs[DEVICE_NR(inode->i_rdev)].shadowname,
00268     &(diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow),
00269     &(diffdevs[DEVICE_NR(inode->i_rdev)].kdev_shadow)) ){
00270     goto fail_sub_open;
00271     return result;
00272 }
00273 printk (<1>Opening device shield (%s)\n",diffdevs[DEVICE_NR(inode->i_rdev)].shieldname);
00274 if ((result=diffdev_try_open (
00275     diffdevs[DEVICE_NR(inode->i_rdev)].shieldname,
00276     &(diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shield),

```

```

00277         &(diffdevs[DEVICE_NR(inode->i_rdev)].kdev_shield) ))){
00278     blkdev_put((diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow), BDEV_RAW);
00279     //bdput(diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow);
00280     goto fail_sub_open;
00281     return result;
00282 }
00283 printk("<1> open successful, let's try to read header\n");
00284 if ((result= init_device(&(diffdevs[DEVICE_NR(inode->i_rdev)]))) )
00285     goto fail_head_init;
00286 else{
00287     diffdev_device *dev=&(diffdevs[DEVICE_NR(inode->i_rdev)]);
00288     diffdev_blksizes[DEVICE_NR(inode->i_rdev)]=dev->head->blksize;
00289     printk("<1>blocksize_size= %d\n",diffdev_blksizes[DEVICE_NR(inode->i_rdev)]);
00290     diffdev_sizes[DEVICE_NR(inode->i_rdev)]=diffdev_blksizes[DEVICE_NR(inode->i_rdev)]*dev->head->blksize;
00291     printk("<1>blocksize= %d\n",diffdev_sizes[DEVICE_NR(inode->i_rdev)]);
00292     diffdev_hardsects[DEVICE_NR(inode->i_rdev)]=hardsect_size[MAJOR(dev->kdev_shadow)]?hardsect_size[MAJO
00293     printk("<1>hardsect= %d\n",diffdev_hardsects[DEVICE_NR(inode->i_rdev)]);
00294 }
00295 return 0;
00296 fail_sub_open:
00297 printk("<1> fail subopen (%i)\n",result);
00298 if (ACQUIRE_CONFIG(DEVICE_NR(inode->i_rdev))){
00299     MOD_DEC_USE_COUNT;
00300     return -ERESTARTSYS;
00301 }
00302 CONFIG_WRITE_LOCK(DEVICE_NR(inode->i_rdev),flags);
00303 diffdevs[DEVICE_NR(inode->i_rdev)].open=0; /*Close device*/
00304 CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00305 RELEASE_CONFIG(DEVICE_NR(inode->i_rdev));
00306 MOD_DEC_USE_COUNT;
00307 return result;
00308 fail_head_init:
00309 if (ACQUIRE_CONFIG(DEVICE_NR(inode->i_rdev))){
00310     MOD_DEC_USE_COUNT;
00311     return -ERESTARTSYS;
00312 }
00313 unlock_subdevice ((diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow));
00314 unlock_subdevice ((diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shield));
00315 CONFIG_WRITE_LOCK(DEVICE_NR(inode->i_rdev),flags);
00316 diffdevs[DEVICE_NR(inode->i_rdev)].open=0; /*Close device*/
00317 CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00318 RELEASE_CONFIG(DEVICE_NR(inode->i_rdev));
00319 MOD_DEC_USE_COUNT;
00320 return result;
00321
00322 }

```

5.1.3.3 int diffdev_release (struct inode * *inode*, struct file * *filp*)

Block device close function.

Check the validity of file being closed. Free private data if needed. We also decrement module usage count here.

Definition at line 328 of file diffmod.c.

```

00329 {
00330     unsigned long flags;

```

```

00331 printk("<1> diffdev_release called\n");
00332 spin_lock_irqsave(&diffdev_lock,flags);
00333 /* configuration device is always openable :) */
00334 if (is_config_device(DEVICE_NR(inode->i_rdev))
00335     {
00336     if (atomic_read(&atome_config)>0)
00337     {
00338     printk("<1> freeing a config diffdev\n");
00339     atomic_dec (&atome_config);
00340     spin_unlock_irqrestore(&diffdev_lock,flags);
00341     MOD_DEC_USE_COUNT;
00342     return 0;
00343     }
00344     else
00345     {
00346     spin_unlock_irqrestore(&diffdev_lock,flags);
00347     printk("<1> Too much release for configure interface");
00348     return 0;
00349     }
00350     }
00351 /*Valid device?*/
00352 printk("<1> validity check\n");
00353 if (!check_diffdev_number(DEVICE_NR(inode->i_rdev))){
00354     spin_unlock_irqrestore(&diffdev_lock,flags);
00355     return -ENODEV;
00356 }
00357 spin_unlock_irqrestore(&diffdev_lock,flags);
00358
00359 ACQUIRE_CONFIG(DEVICE_NR(inode->i_rdev));
00360 CONFIG_WRITE_LOCK(DEVICE_NR(inode->i_rdev),flags);
00361
00362 printk("<1> opened?\n");
00363 if ( !diffdev_is_open(DEVICE_NR(inode->i_rdev)){
00364     CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00365     RELEASE_CONFIG(DEVICE_NR(inode->i_rdev));
00366     return -EPIPE; /*broken pipe error*/
00367 }
00368 CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00369 printk("<1> release shadow?\n");
00370 if (diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow)
00371     {
00372     printk("yes.\n");
00373     blkdev_put((diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow), BDEV_RAW);
00374     //bdput(diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow);
00375     }
00376 printk("<1>release shield?\n");
00377 if (diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shield)
00378     {
00379     printk("yes.\n");
00380     blkdev_put((diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shield), BDEV_RAW);
00381     //bdput(diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shield);
00382     }
00383 save_header (&diffdevs[DEVICE_NR(inode->i_rdev)]);
00384 CONFIG_WRITE_LOCK(DEVICE_NR(inode->i_rdev),flags);
00385 diffdevs[DEVICE_NR(inode->i_rdev)].open=0;
00386 (diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shadow)=NULL;
00387 (diffdevs[DEVICE_NR(inode->i_rdev)].bdev_shield)=NULL;
00388 CONFIG_WRITE_UNLOCK(DEVICE_NR(inode->i_rdev),flags);
00389 RELEASE_CONFIG(DEVICE_NR(inode->i_rdev));
00390 MOD_DEC_USE_COUNT;

```

```
00391 return 0;
00392 }
```

5.1.3.4 int diffdev_ioctl_params (unsigned *command*, unsigned long *argument*)

Mess with device configuration ;-) This function sets or get the device configuration.

It is used when user space programs want to set or get informations about the lower level devices we work on. Setting parameters is only allowed if concerned device is not yet opened and only comming from device number 0!

Definition at line 404 of file diffmod.c.

Referenced by diffdev_ioctl().

```
00405 {
00406     diffdev_params parameters;
00407     unsigned long irqflags;
00408     int devnbr;
00409     char* shadowname;
00410     char* shieldname;
00411     printk("<1> io params for configuration\n");
00412     switch (command)
00413     {
00414
00415         case DIFFDEV_IO_ZEROSHIELD:
00416             printk("reset shield\n");
00417             if (!check_diffdev_number(argument)){
00418                 return -EINVAL;
00419             }
00420             if (ACQUIRE_CONFIG(argument))
00421                 return -ERESTARTSYS;
00422             CONFIG_WRITE_LOCK(argument,irqflags);
00423             SET_FLAG(&(diffdevs[argument]),DIFFFLAG_ZEROSHLD);
00424             CONFIG_WRITE_UNLOCK(argument,irqflags);
00425             RELEASE_CONFIG(argument);
00426             return 0;
00427         case DIFFDEV_IO_GETPARAMS:
00428             printk("<1> io get params\n");
00429             if (copy_from_user (&parameters,(void*)argument,sizeof(diffdev_params))){
00430                 return -EFAULT;
00431             }
00432             devnbr=parameters.device_nbr;
00433             printk("<1>checking device to configure(%i)... \n",devnbr);
00434             if (!check_diffdev_number(devnbr)){
00435                 return -EINVAL;
00436             }
00437             printk("<1>ok\n");
00438             if (ACQUIRE_CONFIG(devnbr))
00439                 return -ERESTARTSYS;
00440             printk("<1> acquired semaphore\n");
00441             parameters.shieldname[0]='\0';
00442             parameters.shadowname[0]='\0';
00443             if (diffdevs[devnbr].shieldname)
00444                 strncpy ((char*)&(parameters.shieldname),diffdevs[devnbr].shieldname,MAX_DEVICE_NAME_LEN);
00445             if (diffdevs[devnbr].shadowname)
```

```

00446         strncpy ((char*)&(parameters.shadowname),diffdevs[devnbr].shadowname,MAX_DEVICE_NAME_LEN);
00447     if (copy_to_user ((void*)argument,&parameters,sizeof(diffdev_params))){
00448         RELEASE_CONFIG(devnbr);
00449         return -EFAULT;
00450     }
00451     printk (<1> sent result\n");
00452     RELEASE_CONFIG(devnbr);
00453     printk (<1> released semaphore\n");
00454     return 0;
00455 case DIFFDEV_IO_SETPARAMS:
00456     if (copy_from_user (&parameters,(void*)argument,sizeof(diffdev_params))){
00457         return -EFAULT;
00458     }
00459     devnbr=parameters.device_nbr;
00460     if (!check_diffdev_number(devnbr)){
00461         return -EINVAL;
00462     }
00463     if (ACQUIRE_CONFIG(devnbr))
00464         return -ERESTARTSYS;
00465
00466     if (diffdev_is_open(devnbr)){
00467         RELEASE_CONFIG(devnbr);
00468         return -EBUSY;
00469     }
00470     if (!valid_parameters (&parameters)){
00471         RELEASE_CONFIG(devnbr);
00472         return -ENOTTY;
00473     }
00474     CONFIG_WRITE_LOCK(devnbr,irqflags);
00475     shadowname=kmalloc (strlen ((char*)&(parameters.shadowname),MAX_DEVICE_NAME_LEN)+1,GFP_KERNEL);
00476     if (!shadowname){
00477         CONFIG_WRITE_UNLOCK(devnbr,irqflags);
00478         RELEASE_CONFIG(devnbr);
00479         return -ENOMEM;
00480     }
00481     strncpy(shadowname,(char*)&(parameters.shadowname),strlen ((char*)&(parameters.shadowname),MAX_
00482     shadowname[strlen ((char*)&(parameters.shadowname),MAX_DEVICE_NAME_LEN)]='\0';
00483     shieldname=kmalloc (strlen ((char*)&(parameters.shieldname),MAX_DEVICE_NAME_LEN)+1,GFP_KERNEL);
00484     if (!shieldname){
00485         kfree (shadowname);
00486         CONFIG_WRITE_UNLOCK(devnbr,irqflags);
00487         RELEASE_CONFIG(devnbr);
00488         return -ENOMEM;
00489     }
00490     strncpy(shieldname,(char*)&(parameters.shieldname),strlen ((char*)&(parameters.shieldname),MAX_
00491     shieldname[strlen ((char*)&(parameters.shieldname),MAX_DEVICE_NAME_LEN)]='\0';
00492     if (diffdevs[devnbr].shadowname)
00493         kfree (diffdevs[devnbr].shadowname);
00494     if (diffdevs[devnbr].shieldname)
00495         kfree (diffdevs[devnbr].shieldname);
00496     diffdevs[devnbr].shadowname=shadowname;
00497     diffdevs[devnbr].shieldname=shieldname;
00498     diffdevs[devnbr].valid=1;
00499     CONFIG_WRITE_UNLOCK(devnbr,irqflags);
00500     RELEASE_CONFIG(devnbr);
00501     return 0;
00502 default:
00503     return -ENOIOCTLCMD;
00504 }
00505 }

```

5.1.3.5 int diffdev_ioctl (struct inode * *inode*, struct file * *filp*, unsigned *command*, unsigned long *argument*)

Ioctl entry point.

This function is responsible for handling ioctl to this device. Valid ioctl commands are:

- DIFFDEV_IO_GETPARAMS
- DIFFDEV_IO_SETPARAMS

Definition at line 512 of file diffmod.c.

```

00513 {
00514     printk (<1>diffdev_ioctl called :)\n");
00515     void *argptr=(void*)argument;
00516     int result;
00517     int device;
00518     long size;
00519     switch (command)
00520     {
00521         case DIFFDEV_IO_GETPARAMS:
00522         case DIFFDEV_IO_SETPARAMS:
00523         case DIFFDEV_IO_ZEROSHIELD:
00524             if (!is_config_device(DEVICE_NR(inode->i_rdev)))
00525                 result = -ENOIOCTLCMD;
00526             else
00527                 result = diffdev_ioctl_params (command,argument);
00528             break;
00529
00530         case DIFFDEV_EMERGENCY:
00531             while (MOD_IN_USE)
00532                 MOD_DEC_USE_COUNT;
00533             MOD_INC_USE_COUNT;
00534             result=0;
00535             break;
00536         case BLKGETSIZE:
00537             if (!argument) return -EINVAL;
00538             if (!access_ok (VERIFY_WRITE,argument,sizeof (long))) return -EFAULT;
00539             device=DEVICE_NR(inode->i_rdev);
00540             size=diffdev_sizes[device];
00541             if (copy_to_user((long*) argument, & size, sizeof (long)))
00542                 return -EFAULT;
00543             return 0;
00544         default:
00545             printk ("parsing to base ioctl function:\n");
00546             printk ("%i,%i\n",MAJOR(inode->i_rdev),MINOR(inode->i_rdev));
00547             return blk_ioctl (inode->i_rdev,command,argument);
00548     }
00549     return result;
00550 }

```

5.1.3.6 int diffdev_checkchange (kdev_t *dev*)

Check media change.

Called by block devices manager to check if media changed. We send true if we are running out of memory

Definition at line 555 of file diffmod.c.

```
00556 {
00557     printk ("<1>check media change\n");
00558     return 0;
00559 }
```

5.1.3.7 int diffdev_revalidate (kdev_t dev)

Revalidate.

similar to media change

Definition at line 563 of file diffmod.c.

```
00564 {
00565     printk ("<1>diffdev_revalidate called\n");
00566     return 0;
00567 }
```

5.1.3.8 int init_module (void)

Initialisation stuff.

This is the main module entry point. It is responsible of initing all stuffs to reference our driver. In order:

- check module loading parameters.
- register the block device operations structure.
- Prepare device specific datas. These are the driver's private datas
- Init static informations related to our device (sector size, block size, device size,...)
- Init a queue to reference our request function

Todo:

FIXME: handling max_sectors according to underlying structure

Definition at line 592 of file diffmod.c.

```
00593 {
00594     int result;
00595     int i;
00596     printk ("<1>Hello, world\n");
00597     spin_lock_init (&diffdev_lock);
00598     atomic_set (&atome_config,0);
00599     if ((major_shadow== -1) || (minor_shadow== -1) || (major_shield== -1) || (minor_shield== -1)){
00600         printk ("<1>Erreur parametres Protecting (%i,%i) with (%i,%i)\n",major_shadow,minor_shadow,major_sh
00601             return -1;
00602     }
```

```

00603 printk("<1>Protecting (%i,%i) with (%i,%i)\n",major_shadow,minor_shadow,major_shield,minor_shield);
00604 diffdev_major = major = DIFFDEV_MAJOR;
00605
00606 /*
00607  * Copy the (static) cfg variables to public prefixed ones to allow
00608  * snoozing with a debugger.
00609  */
00610 diffdev_major = major;
00611 diffdev_devs = devs;
00612 diffdev_rahead = rahead;
00613 diffdev_size = size;
00614 diffdev_blksize = blksize;
00615 diffdev_hardsect = hardsect;
00616
00617 /* 1 register the block device operations structure*/
00618 result = register_blkdev(major, "diffdev", &diffdev_bdops);
00619 if (result <0)
00620 {
00621     printk(KERN_WARNING "diffdev: can't get major number %d\n",diffdev_major);
00622     return result;
00623 }
00624
00625 result = -ENOMEM; /* for the possible errors */
00626 /* 2 Prepare device specific datas */
00627 diffdevs = kmalloc(sizeof(diffdev_device)*diffdev_devs, GFP_KERNEL);
00628 if (!diffdevs)
00629     goto fail_malloc;
00630 for (i=0;i<diffdev_devs;i++)
00631 {
00632     diffdevs[i].open=0;
00633     diffdevs[i].valid=0;
00634     spin_lock_init(&(diffdevs[i].lock));
00635     diffdevs[i].head=NULL;
00636     diffdevs[i].shadowname=NULL;
00637     diffdevs[i].shieldname=NULL;
00638     //diffdevs[i].major_shadow=0;
00639     //diffdevs[i].minor_shadow=0;
00640     //diffdevs[i].major_shield=0;
00641     //diffdevs[i].minor_shield=0;
00642     sema_init (&(diffdevs[i].sem_config),1);
00643     diffdevs[i].rwlock_config=RW_LOCK_UNLOCKED;
00644 }
00645
00646 //sema_init (&sem_config,1);
00647 /* 3 Init static informations related to our device */
00648 read_ahead[major] = diffdev_rahead;
00649 diffdev_sizes = kmalloc(diffdev_devs * sizeof(int), GFP_KERNEL);
00650 if (!diffdev_sizes)
00651     goto fail_malloc;
00652 printk("<1> setting size to %i\n",diffdev_size);
00653 for (i=0; i < diffdev_devs; i++) /* all the same size */
00654     diffdev_sizes[i] = diffdev_size;
00655 blk_size[major]=diffdev_sizes;
00656 diffdev_blksizes = kmalloc(diffdev_devs * sizeof(int), GFP_KERNEL);
00657 if (!diffdev_blksizes)
00658     goto fail_malloc;
00659 for (i=0; i < diffdev_devs; i++) /* all the same blocksize */
00660     diffdev_blksizes[i] = diffdev_blksize;
00661 blksize_size[major]=diffdev_blksizes;
00662 diffdev_hardsects = kmalloc(diffdev_devs * sizeof(int), GFP_KERNEL);

```

```

00663 if (!diffdev_hardsects)
00664     goto fail_malloc;
00665 for (i=0; i < diffdev_devs; i++) /* all the same hardsect */
00666     diffdev_hardsects[i] = diffdev_hardsect;
00667 hardsect_size[major]=diffdev_hardsects;
00668 /* FIXME: handling max_sectors according to underlying structure*/
00669
00670
00671 //shadow=bdget(MKDEV(major_shadow,minor_shadow));
00672 //result=blkdev_get (shadow,FMODE_WRITE,0,BDEV_RAW);
00673 //printk ("<1> error in shdow get: %i\n",result);
00674 /* 4 Init a queue to reference our request function */
00675 printk ("<1> Initing a queue for %i at %p\n",diffdev_major,BLK_DEFAULT_QUEUE(diffdev_major));
00676 printk ("<1> Managing it's own request function :)\n");
00677 //blk_init_queue (BLK_DEFAULT_QUEUE(diffdev_major), diffdev_request);
00678 blk_queue_make_request (BLK_DEFAULT_QUEUE(diffdev_major),diffdev_make_request);
00679 return 0;
00680
00681 fail_malloc:
00682 if (diffdevs) kfree (diffdevs);
00683 diffdevs=NULL;
00684 read_ahead[major] = 0;
00685 if (diffdev_sizes) kfree(diffdev_sizes);
00686 blk_size[major] = NULL;
00687 if (diffdev_blksizes) kfree(diffdev_blksizes);
00688 blksize_size[major] = NULL;
00689 if (diffdev_hardsects) kfree(diffdev_hardsects);
00690 hardsect_size[major] = NULL;
00691 unregister_blkdev(major,"diffdev");
00692 return result;
00693
00694 }

```

5.1.3.9 void cleanup_module (void)

Module exit and cleanup point.

Here we clean all datas associated with our driver. Most of them were initied in `init_module()` (p. 23).

Todo:

FIXME: support for max_readahead and max_sectors

Definition at line 700 of file diffmod.c.

```

00701 {
00702     int i;
00703     printk ("<1>Goodbye cruel world\n");
00704     printk ("<1>fsyncing devices(%d)\n",diffdev_devs);
00705     for (i=0; i<diffdev_devs; i++)
00706         fsync_dev(MKDEV(diffdev_major, i)); /* flush the devices */
00707     printk ("<1>Unregistering device major number\n");
00708     unregister_blkdev(major, "diffdev");
00709     printk ("<1> cleanup default queue for %i: %p\n",diffdev_major,BLK_DEFAULT_QUEUE(diffdev_major));
00710     //blk_cleanup_queue (BLK_DEFAULT_QUEUE(diffdev_major));
00711     for (i=0;i<diffdev_devs;i++)
00712     {

```

```

00713     if (diffdevs[i].open)
00714         printk ("<1> Alert in diffdev cleanup. Usage count is zero but a device (%i) is still open\n",i);
00715     if (diffdevs[i].shadowname)
00716         kfree (diffdevs[i].shadowname);
00717     if (diffdevs[i].shieldname)
00718         kfree (diffdevs[i].shieldname);
00719 }
00720 kfree (diffdevs);
00721 /* Clean up the global arrays */
00722 printk ("<1>readahead[major]=0\n");
00723 read_ahead[major] = 0;
00724 printk ("<1>kfree(blk_size[major])\n");
00725 kfree(blk_size[major]);
00726 blk_size[major] = NULL;
00727 printk ("<1>kfree(blksize_size[major])\n");
00728 kfree(blksize_size[major]);
00729 blksize_size[major] = NULL;
00730 printk ("<1>kfree(hardsect_size[major])\n");
00731 kfree(hardsect_size[major]);
00732 hardsect_size[major] = NULL;
00733 /* FIXME: max_readahead and max_sectors */
00734 }

```

5.1.4 Variable Documentation

5.1.4.1 diffdev_device* diffdevs = NULL

diffdev_device (p. 2) array.

Contains an array with all datas needed for handling each device of the
Definition at line 131 of file diffmod.c.

5.1.4.2 struct block_device_operations diffdev_bdops

Initial value:

```

{
    open:                diffdev_open,
    release:             diffdev_release,
    ioctl:               diffdev_ioctl,
    check_media_change:  diffdev_checkchange,
    revalidate:          diffdev_revalidate,
}

```

Block devide operations.

This is the structure we pass to the kernel to reference our driver. All fileds are self explaining.

Definition at line 572 of file diffmod.c.

5.2 diffmod.h File Reference

structures and macros definition.

Data Structures

- struct **diffdev_params**
Ioctl params structure.
- struct **diffdev_table_entry**
The following structure is used to define a 'shield' for a sector.
- struct **diffdev_superblock_entry**
shield super block.
- struct **diffdev_entry**
diffdev physical device specific datas.
- struct **diffdev_device**
Virtual device datas.
- struct **diffdev_action**
- struct **superblock_w_op**
super block saving structure.
- struct **superblock_op**
Used to delay a sub operation.
- struct **sub_device_op**
Used to keep track of a sub operation.

Defines

- #define **MAX_DEVICE_NAME_LEN** 256
- #define **DIFFDEV_IOCTL_BASE** 0xF0
IOCTL base command for diffdev driver.
- #define **DIFFDEV_EMERGENCY_IO**(DIFFDEV_IOCTL_BASE,0)
- #define **DIFFDEV_IO_GETPARAMS** _IOR(DIFFDEV_IOCTL_BASE,0,diffdev_params)
ioctl command: get device configuration parameters.
- #define **DIFFDEV_IO_SETPARAMS** _IOW(DIFFDEV_IOCTL_BASE,0,diffdev_params)
ioctl command: get device configuration parameters.
- #define **DIFFDEV_IO_ZEROSHIELD** _IOW(DIFFDEV_IOCTL_BASE,1,unsigned long)
ioctl command: zero shield.

- #define **DIFFDEV_MAILTO** printk ("<1>[diffdev]Mail a bug report to tchize@mailandnews.com !!\n");BUG()
- #define **DIFFINFO**(format, arg...)
- #define **get_defaultize**(_table, _x, _default) (_table?(_table[_x]):(_default))
- #define **DIFFDEV_MAJOR** 240
static developpement major number by default (0xF0).
- #define **DIFFDEV_DEVS** 256
256 disks.
- #define **DIFFDEV_RAHEAD** 2
two sectors readhead.
- #define **DIFFDEV_SIZE** 1024
0 size by default.
- #define **DIFFDEV_BLKSIZE** 4096
1k blocks.
- #define **DIFFDEV_HARDSECT** 512
2.2 and 2.4 can use different values for sector size. We keep 512.
- #define **offsetof**(type_, member_) ((u32)&(((type_*)0) → member_ -))
gets the offset of a member inside a structure.
- #define **DIFFFLAG_NONE** 0x00000000
No flag defined.
- #define **DIFFFLAG_ZEROSHLD** 0x00000001
Flag reset(create) shield on mount.
- #define **DIFFFLAG_NOPE** 0x00000002
- #define **DIFFFLAG_NOPE** 0x00000004
- #define **DIFFFLAG_NOPE** 0x00000008
- #define **DIFFFLAG_NOPE** 0x00000010
- #define **DIFFFLAG_NOPE** 0x00000020
- #define **DIFFFLAG_NOPE** 0x00000040
- #define **DIFFFLAG_NOPE** 0x00000080
- #define **TEST_FLAG**(_diffdevdev, _diffdevflag) (((_diffdevdev) → flags) & (_diffdevflag))
- #define **SET_FLAG**(_diffdevdev, _diffdevflag) (((_diffdevdev) → flags) |= (_diffdevflag))
- #define **CLEAR_FLAG**(_diffdevdev, _diffdevflag) (((_diffdevdev) → flags) &= (!(_diffdevflag)))

- #define **LOCK_NONE** 0
- #define **LOCK_SIMPLE** 1
- #define **LOCK_IRQ** 2
- #define **MAGIC_DIFFDEV_BASE** 0x2044666
Basic diffdev magicnumber.
- #define **SET_MAGIC**(_ptr, _magic) _ptr → magic=_magic
*set the magic number of a struct *.*
- #define **NEED_MAGIC**(_ptr, _magic)
*Need the presence of a magic number in a struct *.*
- #define **MAGIC_DIFFDEV_ENTRY** MAGIC_DIFFDEV_BASE+1
magic number for a diffdev_entry (p. 3) structure.
- #define **MAGIC_DIFFDEV_SUBOP** MAGIC_DIFFDEV_BASE+2
magic number for a sub_device_op (p. 7) structure.
- #define **MAGIC_DIFFDEV_SBLKOP** MAGIC_DIFFDEV_BASE+3
magic number for a superblock_op (p. 8) structure.
- #define **MAGIC_DIFFDEV_SBLKW** MAGIC_DIFFDEV_BASE+4
magic number for a superblock_w_op (p. 9) structure.
- #define **SB>Loading** 1
- #define **SB>Saving** 2
- #define **SB_NeedSave** 3
- #define **SB_opswaiting** 4

Variables

- **superblock_op superblock_op**
Used to delay a sub operation.

5.2.1 Detailed Description

structures and macros definition.

Definition and macros in this file are protected so this header can be used for userspace program and for kernel space driver. For the protection, it check the definition for the macro `__KERNEL__`

Definition in file **diffmod.h**.

5.2.2 Define Documentation

5.2.2.1 `#define DIFFDEV_IO_ZEROSHIELD _IOW(DIFFDEV_IOCTL_BASE,1,unsigned long)`

ioctl command: zero shield.

Create a new root shield on mount. Used when creating a new device

Definition at line 55 of file diffmod.h.

5.2.2.2 `#define DIFFDEV_SIZE 1024`

0 size by default.

Todo:

FIXME size depend on lower disk size.

Definition at line 85 of file diffmod.h.

5.2.2.3 `#define offsetof(type--, member--) ((__u32)(amp(((type--*)0) -> member--)))`

gets the offset of a member inside a structure.

It maps the structure to a null pointer and gets the address of the member

Definition at line 92 of file diffmod.h.

Referenced by `allocate_header()`, `init_device_from_scratch()`, and `load_device_from_disk()`.

5.2.2.4 `#define MAGIC_DIFFDEV_BASE 0x2044666`

Basic diffdev magicnumber.

value is "double devil". If you don't know why, mail me: tchize@mailandnews.com

Using magic numbers is recommended to detect "memory trashing or polluting" in kernel structures. see linux doc in magic-number.txt

Definition at line 154 of file diffmod.h.

5.2.2.5 `#define NEED_MAGIC(_ptr, _magic)`

Value:

```
if (_ptr->magic!=_magic) {\
    printk (KERN_EMERG __FILE__ " in " __FUNCTION__ "(): invalid magic: 0x%X should be 0x%X\n",_ptr->magic,_magic)
```

Need the presence of a magic number in a struct *.

If this is not the case, there has been memory corruption. we issue an emergency message to the kernel and syslogd

Definition at line 161 of file diffmod.h.

Referenced by `end_block_operation()`, `end_stub_operation()`, and `end_sub_operation()`.

5.2.2.6 `#define MAGIC_DIFFDEV_ENTRY MAGIC_DIFFDEV_BASE+1`

magic number for a `diffdev_entry` (p. 3) structure.

Base magic + 1.

Definition at line 166 of file diffmod.h.

5.2.2.7 `#define MAGIC_DIFFDEV_SUBOP MAGIC_DIFFDEV_BASE+2`

magic number for a `sub_device_op` (p. 7) structure.

Base magic + 2

Definition at line 170 of file diffmod.h.

5.2.2.8 `#define MAGIC_DIFFDEV_SBLKOP MAGIC_DIFFDEV_BASE+3`

magic number for a `superblock_op` (p. 8) structure.

Base magic + 3

Definition at line 174 of file diffmod.h.

5.2.2.9 `#define MAGIC_DIFFDEV_SBLKW MAGIC_DIFFDEV_BASE+4`

magic number for a `superblock_w_op` (p. 9) structure.

Base magic + 4

Definition at line 178 of file diffmod.h.

5.2.3 Variable Documentation

5.2.3.1 `struct superblock_op superblock_op`

Used to delay a sub operation.

This structure is read when a superblock has finished loading. this is a linked list of awaiting orders!

5.3 difftransfert.c File Reference

Transfert toolkit.

```
#include <linux/autoconf.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/blkdev.h>
#include <linux/slab.h>
#include "diffmod.h"
#include "diffproto.h"
```

Defines

- #define **MODULE**
Needed to create a module.
- #define **DEBUG**

Functions

- unsigned long **block2sector** (unsigned long block, **diffdev_entry** *device)
convert a block number to a sector number.
- unsigned long **sector2block** (unsigned long sector, **diffdev_entry** *device)
convert a sector number to a block number.
- void **end_block_operation** (struct buffer_head *bh, int uptodate)
this the callback used to notify a block op is finished.
- void **end_stub_operation** (struct buffer_head *bh, int uptodate)
simply be the stop point of a block operation.
- void **end_sub_operation** (struct buffer_head *bh, int uptodate)
this the callback used to notify a subdevice op is finished.
- int **read_superblock** (struct buffer_head *bh, **diffdev_device** *device, __u32 superblock_nr, int direction)
check availability of super block.
- int **prepare_superblock** (struct buffer_head *bh, **diffdev_device** *device, __u32 superblock_nr)
Perpare a superblock due to a write operation.
- void **finish_superblock** (**diffdev_device** *device, __u32 super)

save a superblock to disk.

- void **rw_block_physical** (`_u32 block_nr`, `struct buffer_head *bh`, `diffdev_device *dev`, `int direction`, `kdev_t realdevice`)
read/write a block to a sub device.
- void **rw_block_shield** (`_u32 block_nr`, `struct buffer_head *bh`, `diffdev_device *dev`, `int direction`)
read/write a block to shield device.
- void **rw_block_shadow** (`_u32 block_nr`, `struct buffer_head *bh`, `diffdev_device *dev`, `int direction`)
read/write a block to shadow device.
- void **try_read_block** (`struct buffer_head *bh`, `diffdev_device *device`)
Tries to read a block from diffdev.
- void **try_write_block** (`struct buffer_head *bh`, `diffdev_device *device`)

5.3.1 Detailed Description

Transfert toolkit.

This file contain all functions needed to handle block device transferts, and repartition.

Definition in file **difftransfert.c**.

5.3.2 Function Documentation

5.3.2.1 unsigned long block2sector (unsigned long *block*, diffdev_entry * *device*) [inline]

convert a block number to a sector number.

Since the diffdev and it's underlying devices must have the same `hardsect_size` and `blksize_size`, this informations is kept in `device->dev` structure. We use those to calculate transitions

Definition at line 46 of file `difftransfert.c`.

Referenced by `rw_block_physical()`.

```
00047 {
00048     return block*(device->blksize/device->hardsect_size);
00049 }
```

5.3.2.2 unsigned long sector2block (unsigned long *sector*, diffdev_entry * *device*) [inline]

convert a sector number to a block number.

Since the diffdev and it's underlying devices must have the same `hardsect_size` and `blksize_size`, this informations is kept in `device->dev` structure. We use those to calclate trasnitions

Definition at line 56 of file `difftransfert.c`.

Referenced by `try_read_block()`.

```
00057 {
00058     return sector/(device->blksize/device->hardsect_size);
00059 }
```

5.3.2.3 void end_block_operation (struct buffer_head * bh, int uptodate)

this the callback used to notify a block op is finished.

we read the private field which is of type `superblock_op` (p.8) and read informations in it to restart the transfert the diffdev caller asked. Note that we check the magic.

Definition at line 65 of file `difftransfert.c`.

```
00066 {
00067     superblock_op *operation=(superblock_op*)bh->b_private;
00068     superblock_op *tmp;
00069     diffdev_entry* dev;
00070     if ((operation==NULL))
00071     {
00072         BUG();
00073         return;
00074     }
00075     spin_lock (operation->device->head->lock);
00076     /* reread since this can have changed while we were waiting for the lock */
00077     operation=(superblock_op*)bh->b_private;
00078     NEED_MAGIC (operation,MAGIC_DIFFDEV_SBLKOP);
00079     dev=operation->device->head;
00080     bh->b_private=NULL;
00081     if (uptodate)
00082     {
00083         /*the block should be a pointer to dev->superblock_size*sizeof(diffdev_table_entry) but
00084         it is never bigger than a block and should be rounded to a block for direct mapping.*/
00085         dev->superblock[operation->superblock_nr].block=kmalloc (dev->blksize,GFP_ATOMIC);
00086         memcpy (dev->superblock[operation->superblock_nr].block,bh->b_data,bh->b_size);
00087     }
00088     mark_buffer_uptodate(bh, uptodate);
00089     unlock_buffer(bh);
00090     brelse (bh);
00091     spin_unlock (operation->device->head->lock);
00092     /*Go through the list of awaiting operations*/
00093     while (operation)
00094     {
00095         if (!uptodate) /*simply cancel operation*/
00096             operation->request->b_end_io(operation->request,uptodate);
00097     }
```

```

00098     else
00099     {
00100     /* superblock entry is now ready. Let's retry the diffdev operation*/
00101     if (operation->op_type==WRITE)
00102         try_write_block (operation->request,operation->device);
00103     else
00104         try_read_block (operation->request,operation->device);
00105     }
00106     tmp=operation->next;
00107     kfree (operation);
00108     operation=tmp;
00109 }
00110
00111 }

```

5.3.2.4 void end_stub_operation (struct buffer_head * bh, int upto-date)

simply be the stop point of a block operation.

This function does nothing else than unlock bh. It is mainly used by `finnish_superblock`

Definition at line 116 of file `difftransfert.c`.

```

00117 {
00118     superblock_w_op* wop;
00119     wop=(superblock_w_op*)bh->b_private;
00120     if (!wop)
00121     {
00122         DIFFINFO ("---> no wop, abording\n");
00123         return;
00124     }
00125     NEED_MAGIC (wop,MAGIC_DIFFDEV_SBLKW);
00126     if (!uptodate)
00127         DIFFINFO ("a sorry, we couldn't save :(\n");
00128     spin_lock (&wop->device->head->lock);
00129     clear_bit (SB_Saving,&(wop->device->head->superflags[wop->superblock_nr]));
00130     if (test_and_clear_bit (SB_NeedSave,&(wop->device->head->superflags[wop->superblock_nr])))
00131     {
00132         /*NeedSave bit at 1. So, let's go: extra ball*/
00133         memcpy (bh->b_data,wop->device->head->superblock[wop->superblock_nr].block,wop->device->head->blksize);
00134         DIFFINFO ("resaving superblock once again");
00135         spin_unlock (&wop->device->head->lock);
00136         submit_bh (WRITE,bh);
00137     }
00138     else
00139     {
00140         mark_buffer_uptodate(bh, uptodate);
00141         unlock_buffer(bh);
00142         spin_unlock (&wop->device->head->lock);
00143         brelse (bh);
00144         kfree (wop);
00145         DIFFINFO ("done stub operation\n");
00146     }
00147 }

```

5.3.2.5 void end_sub_operation (struct buffer_head * bh, int uptodate)

this the callback used to notify a subdevice op is finished.

we read the private field which is of type **sub_device_op** (p. 7) and read informations in it to notify an end of transfert to the diffdev caller. Note that we check the magic.

Definition at line 153 of file difftransfert.c.

```

00154 {
00155     sub_device_op *operation=(sub_device_op*)bh->b_private;
00156     NEED_MAGIC (operation,MAGIC_DIFFDEV_SUBOP);
00157     DIFFINFO (__FUNCTION__ " subdeviceop finnished\n");
00158     if ((operation==NULL))
00159         {
00160             BUG();
00161             return;
00162         }
00163     if ((uptodate) && ((operation->op_type==READ)|| (operation->op_type==READA)))
00164     {
00165         memcpy (operation->request->b_data,bh->b_data,operation->request->b_size);
00166     }
00167     mark_buffer_uptodate(bh, uptodate);
00168     unlock_buffer(bh);
00169     brelse (bh);
00170     operation->request->b_end_io(operation->request,uptodate);
00171     kfree (operation);
00172
00173 }
```

5.3.2.6 int read_superblock (struct buffer_head * bh, diffdev_device * device, __u32 superblock_nr, int direction)

check availability of super block.

if superblock exists and is not in memory, load it This function is meant only for read block availability, so it does not allocate a new block to do write operation. We are supposed to hold device->head->lock !

Definition at line 180 of file difftransfert.c.

Referenced by prepare_superblock(), and try_read_block().

```

00181 {
00182     diffdev_entry* dev=device->head;
00183     spin_lock (&device->head->lock);
00184     if (
00185         (dev->superblock[superblock_nr].block)||
00186         (dev->superblock_addr[superblock_nr]==0)
00187     )
00188     {
00189         spin_unlock (&device->head->lock);
00190         return 1; /*Already in memory or does not exist*/
00191     }
00192     else
```

```

00193 {
00194     /*a superbloc exist and it is not in memory.
00195         load from disk and delay current transfert by returning 0*/
00196
00197     DIFFINFO (__FUNCTION__ " Delaying read: superbloc needed(%i)\n",dev->superblock_addr[superblock_nr])
00198     superbloc_op* sop=kmalloc (sizeof(superblock_op),GFP_ATOMIC);
00199     struct buffer_head* sbh;
00200     SET_MAGIC (sop,MAGIC_DIFFDEV_SBLKOP);
00201     sop->request=bh;
00202     sop->op_type=direction;
00203     sop->device=device;
00204     sop->superblock_nr=superblock_nr;
00205     sbh=getblk (device->kdev_shield,dev->superblock_addr[superblock_nr],dev->blksize);
00206     if (test_and_set_bit(BH_Lock, &sbh->b_state))
00207     {
00208         /*Somebody is awaiting it. Since we locked the device, this can be only us.
00209             If this is not the case, we are running into trouble :(*/
00210         sop->next=(superblock_op*)sbh->b_private;
00211         sbh->b_private=sop;
00212         spin_unlock (&device->head->lock);
00213     }
00214     else
00215     {
00216         sop->next=NULL;
00217         sbh->b_end_io=end_block_operation;
00218         sbh->b_private=sop;
00219         spin_unlock (&device->head->lock);
00220         submit_bh (READ,sbh);
00221     }
00222     return 0;
00223 }
00224 }

```

5.3.2.7 int prepare_superblock (struct buffer_head * bh, diffdev_ device * device, __u32 superbloc_nr)

Perpare a superbloc due to a write operation.

if there is already a physical block associated with this device then call read_superblock else create it and allocate it a physical area. We are supposed to hold device->head->lock !

Definition at line 230 of file difftransfert.c.

```

00231 {
00232     __u32 i;
00233     diffdev_entry* dev=device->head;
00234     if (dev->superblock_addr[superblock_nr]>0) /*there is a physical address. simple ensure bolck is in mem
00235         return read_superblock(bh,device,superblock_nr,WRITE);/*Be sure to load it if it already exists*/
00236     else
00237     {
00238         dev->superblock[superblock_nr].block=(diffdev_table_entry*)kmalloc (dev->blksize,GFP_ATOMIC);
00239         for (i=0;i<dev->superblock_size;i++)
00240         {
00241             dev->superblock[superblock_nr].block[i].COR=0;
00242             dev->superblock[superblock_nr].block[i].MOD=0;
00243             dev->superblock[superblock_nr].block[i].shield=0;
00244         }

```

```

00245     dev->superblock_addr[superblock_nr]=get_free_block(dev,LOCK_NONE);
00246     return 1;
00247 }
00248 }

```

5.3.2.8 void finnish_superblock (diffdev_device * device, __u32 super)

save a superblock to disk.

this function tries to save a superblock to disk.

Definition at line 252 of file difftransfert.c.

```

00253 {
00254     diffdev_entry* dev=device->head;
00255     struct buffer_head *sbh;
00256     int result;
00257     if (dev->superblock[super].block==NULL)
00258     {
00259         DIFFINFO ("Asked to finnish a superblock(%d) but it has no blocks!!!\n",super);
00260         return; /*Nothing to save*/
00261     }
00262
00263     if (dev->superblock_addr[super]==0) /*Allocate a block for datas*/
00264     {
00265         DIFFINFO ("allocating address\n");
00266         dev->superblock_addr[super]=get_free_block(dev,LOCK_NONE);
00267         dev->needsave=1;
00268     }
00269     //DIFFINFO ("sorry, saves of superblocks disabled...\n");
00270     //return;
00271     spin_lock (&device->head->lock);
00272     result = test_and_set_bit (SB_Saving,&(dev->superflags[super]));
00273     if (result)
00274     {
00275         set_bit (SB_NeedSave,&(dev->superflags[super]));
00276         spin_unlock (&device->head->lock);
00277     }
00278     else
00279     {
00280         superblock_w_op* wop;
00281         spin_unlock (&device->head->lock);
00282         wop=(superblock_w_op*)kmalloc (sizeof (superblock_w_op),GFP_ATOMIC);
00283         SET_MAGIC (wop,MAGIC_DIFFDEV_SBLKW);
00284         wop->device=device;
00285         wop->superblock_nr=super;
00286         DIFFINFO ("finishing superblock %d by writing at block %d\n",super,dev->superblock_addr[super]);
00287         sbh=getblk (device->kdev_shield, dev->superblock_addr[super], dev->blksize);
00288         memcpy (sbh->b_data,dev->superblock[super].block,dev->blksize);
00289         sbh->b_end_io=end_stub_operation;
00290         sbh->b_private=wop;
00291         DIFFINFO (__FUNCTION__ " buffer_head being submitted for finnish block :)\n");
00292         if (test_and_set_bit(BH_Lock, &sbh->b_state))
00293             DIFFINFO ("-----> several saves at the same time! <-----");
00294         else
00295             submit_bh (WRITE,sbh);
00296     }

```

```
00297 }
```

5.3.2.9 void rw_block_physical (__u32 block_nr, struct buffer_head * bh, diffdev_device * dev, int direction, kdev_t realdevice)

read/write a block to a sub device.

This function translates block numbers to sectors, creates a buffer head for the transfer and asks for the transfer. No warranty is given of time transfert completes.

Definition at line 305 of file difftransfert.c.

Referenced by rw_block_shadow(), and rw_block_shield().

```
00306 {
00307     unsigned long sector;
00308     sub_device_op* subop;
00309     struct buffer_head *sbh;
00310     //bh->b_end_io (bh,0);/*impossible to read. device not ready*/
00311     //DIFFINFO (__FUNCTION__ " Stop here!\n");
00312     //return;
00313     sector=block2sector (block_nr, dev->head);
00314     sbh=getblk (realdevice, block_nr, bh->b_size);
00315     //if (direction==WRITE)
00316     //{
00317     // memcpy (sbh->b_data,bh->b_data,bh->b_size);
00318     // mark_buffer_dirty (bh);
00319     // brelse (sbh);
00320     // bh->b_end_io(bh,1);
00321     // return;
00322     //}
00323     subop=kmalloc (sizeof(sub_device_op),GFP_ATOMIC);
00324     SET_MAGIC (subop,MAGIC_DIFFDEV_SUBOP);
00325     subop->try_count=3; /*try max 3 time the operation*/
00326     subop->request=bh;
00327     subop->op_type=direction;
00328     subop->device=dev;
00329     DIFFINFO (__FUNCTION__ " buffer_head being submitted :)\n");
00330     if (test_and_set_bit(BH_Lock, &sbh->b_state))
00331         //BUG();
00332         {
00333             DIFFINFO ("read canceled already reading\n");brelse (bh);
00334             subop->request->b_end_io(subop->request,0);
00335             kfree (subop);
00336             brelse (sbh);
00337             return;
00338         }
00339     if (direction==WRITE)
00340         memcpy (sbh->b_data,bh->b_data,bh->b_size);
00341     sbh->b_end_io=end_sub_operation;
00342     sbh->b_private=subop;
00343     submit_bh (direction,sbh);
00344 }
```

5.3.2.10 void rw_block_shield (__u32 block_nr, struct buffer_head * bh, diffdev_device * dev, int direction)

read/write a block to shield device.

this function uses `rw_block_physical` to do his job.

Definition at line 348 of file `difftransfert.c`.

Referenced by `try_read_block()`.

```
00349 {
00350     DIFFINFO (KERN_ALERT "Reading/writing from shield\n");
00351     rw_block_physical (block_nr,bh,dev,direction,dev->kdev_shield);
00352 }
```

5.3.2.11 void `rw_block_shadow` (`__u32 block_nr`, `struct buffer_head * bh`, `diffdev_device * dev`, `int direction`)

read/write a block to shadow device.

this function uses `rw_block_physical` to do his job.

Definition at line 357 of file `difftransfert.c`.

Referenced by `try_read_block()`.

```
00358 {
00359     DIFFINFO ("Reading/writing from shadow\n");
00360     rw_block_physical (block_nr,bh,dev,direction,dev->kdev_shadow);
00361 }
```

5.3.2.12 void `try_read_block` (`struct buffer_head * bh`, `diffdev_device * device`)

Tries to read a block from `diffdev`.

If informations about superblock are enough for now, they are used immediatly to read the block. If not, the read is delayed, a command is issued to read superblock and the function return. the `b_end_io` callback of the superblock reading operation will take care of recalling this function when datas are available.

Definition at line 370 of file `difftransfert.c`.

Referenced by `diffdev_make_request()`, and `end_block_operation()`.

```
00371 {
00372     __u32 superblock_nr;
00373     __u32 subblock_idx;
00374     __u32 block_nr;
00375     diffdev_entry *dev;
00376     dev=device->head;
00377     if (!dev)
00378     {
00379         bh->b_end_io (bh,0);/*impossible to read. device not ready*/
00380         DIFFINFO (__FUNCTION__ " No device to read from\n");
00381         return;
00382     }
```

```

00382     }
00383     dump_device (dev);
00384     block_nr=sector2block(bh->b_rsector,device->head);
00385     DIFFINFO ("sector2block (%d)=%d\n",bh->b_rsector,block_nr);
00386     superblock_nr=block_nr/device->head->superblock_size;
00387     subblock_idx=block_nr%device->head->superblock_size;
00388     if (!read_superblock (bh,device,superblock_nr,READ))
00389     {
00390         /*delay reading and ask for superblock reading*/
00391         DIFFINFO (__FUNCTION__ " delaying read\n");
00392         return;
00393     }
00394     //if ((device->head->superblock[superblock_nr].block) &&
00395     //    (device->head->superblock[superblock_nr].block[subblock_idx].MOD))
00396     //    rw_block_shield (device->head->superblock[superblock_nr].block[subblock_idx].shield,bh,device,READ)
00397     //else
00398     //    rw_block_shadow (block_nr,bh,device,READ);
00399     if (!device->head->superblock)
00400     {
00401         DIFFINFO ("warning there's no superblock inited :(((");
00402         bh->b_end_io (bh,0);/*impossible to read. device not ready*/
00403         DIFFINFO (__FUNCTION__ " Stop here!\n");
00404         return;
00405     }
00406     if (device->head->superblock[superblock_nr].block)
00407     {
00408         DIFFINFO ("head says there is a block in memory (%p)\n",device->head->superblock[superblock_nr].blo
00409         //bh->b_end_io (bh,0);/*impossible to read. device not ready*/
00410         //DIFFINFO ("we should read from %d\n",device->head->superblock[superblock_nr].block[subblock_idx].
00411         //DIFFINFO (__FUNCTION__ " Stop here!\n");
00412         //return;
00413         if (device->head->superblock[superblock_nr].block[subblock_idx].MOD)
00414         {
00415             rw_block_shield (device->head->superblock[superblock_nr].block[subblock_idx].shield,bh,device,RE
00416             return;
00417         }
00418     }
00419     // bh->b_end_io (bh,0);/*impossible to read. device not ready*/
00420     // DIFFINFO (__FUNCTION__ " Stop here(call on shadow)!\n");
00421     // return;
00422     rw_block_shadow (block_nr,bh,device,READ);
00423 }

```

5.4 diffutil.c File Reference

Device toolkit.

```

#include <linux/autoconf.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/blkdev.h>
#include <linux/slab.h>
#include "diffmod.h"

```

```
#include "diffproto.h"
```

Defines

- **#define MODULE**
Needed to create a module.

Functions

- int **diffdev_try_open** (char *dev_name, struct block_device **bdevptr, kdev_t *kdevptr)
Tries to open device called devname.
- **_u32 get_free_block** (diffdev_entry *dev, int lockmode)
allocate a free block on a shield device.
- int **lock_subdevice** (struct inode *inode, mode_t mode, struct block_device **devptr)
Lock a device to prevent concurrent access.
- int **unlock_subdevice** (struct block_device *dev)
Unlock a device previously locked.
- int **compatible_devices** (diffdev_device *dev)
check compatibility of shield and shadow.
- void **save_header** (diffdev_device *dev)
- **diffdev_entry * allocate_header** (diffdev_device *dev)
allocate a header.
- void **dump_device** (diffdev_entry *dev)
- int **load_device_from_disk** (diffdev_device *dev)
create a diffdev header from disk.
- int **init_device_from_scratch** (diffdev_device *dev)
create a diffdev header from scratch.
- int **init_device** (diffdev_device *dev)
Init a diffdev device.
- int **is_valid_request** (struct buffer_head *bh, diffdev_device *device)

5.4.1 Detailed Description

Device toolkit.

This file contain all functions needed to handle block device configuration and locking.

Definition in file `diffutil.c`.

5.4.2 Function Documentation

5.4.2.1 `int diffdev_try_open (char * dev_name, struct block_device ** bdevptr, kdev_t * kdevptr)`

Tries to open device called devname.

devname is a filename used to acces a device. Most of this code check for existenz (seen the movie?) of this special file and for the fact it's a free block device. Most of this code was ripped from `get_sb_bdev` used by mount functions. Since this code was ripped from a function where a process context was supposed to exist, it possible this function sleeps. So try not to call with any spinlock hold!

Definition at line 48 of file `diffutil.c`.

Referenced by `diffdev_open()`.

```

00049 {
00050     struct inode *inode;
00051     int error = 0;
00052     mode_t mode = FMODE_READ|FMODE_WRITE; /* we always need it ;-) */
00053     struct nameidata nd;
00054     printk ("<1>trying to open %s\n",dev_name);
00055     if (!dev_name || !*dev_name)
00056         return -EINVAL;
00057     if (path_init(dev_name, LOOKUP_FOLLOW|LOOKUP_POSITIVE, &nd))
00058         error = path_walk(dev_name, &nd);
00059     if (error)
00060         return (error);
00061     inode = nd.dentry->d_inode;
00062     error = -EINVAL;
00063     error = -EACCES;
00064     if (nd.mnt->mnt_flags & MNT_NODEV)
00065         goto sub_open_release;
00066     printk ("<1>device: %d:%d\n",MAJOR(inode->i_rdev),MINOR(inode->i_rdev));
00067
00068     if ((error=lock_subdevice (inode, mode, bdevptr)))
00069         goto sub_open_release;
00070     error=0;
00071     (*kdevptr)= inode->i_rdev;
00072 sub_open_release:
00073     printk ("<1>releasing path\n");
00074     path_release (&nd);
00075     return error;
00076
00077 }
```

5.4.2.2 __u32 get_free_block (diffdev_entry * dev, int lockmode)

allocate a free block on a shield device.

The needlock parameter is used to know if get_free_block has to take hold of the device lock.

Todo:

- When running out of free block, wakeup any awaiting daemon.
- Really out of free block: go to emergency read only

Definition at line 84 of file diffutil.c.

Referenced by `finnish_superblock()`, `init_device_from_scratch()`, and `prepare_superblock()`.

```

00085 {
00086     unsigned long flags;
00087     __u32 block;
00088     if (lockmode!=LOCK_NONE)
00089     {
00090         if (lockmode==LOCK_SIMPLE)
00091             spin_lock(&dev->lock);
00092         if (lockmode==LOCK_IRQ)
00093             spin_lock_irqsave(&dev->lock,flags);
00094     }
00095     if (dev->current_free_block>=dev->nr_block_shield)
00096     {
00097         printk (<1>*** Aleeerrt, No blocks available for a one block transaction! ***\n");
00098         block= dev->nr_block_shield-1;
00099     }
00100     else
00101         block = dev->current_free_block++;
00102
00103     if (lockmode!=LOCK_NONE)
00104     {
00105         if (lockmode==LOCK_SIMPLE)
00106             spin_unlock(&dev->lock);
00107         if (lockmode==LOCK_IRQ)
00108             spin_unlock_irqrestore(&dev->lock,flags);
00109     }
00110     return block;
00111 }

```

5.4.2.3 int lock_subdevice (struct inode * inode, mode_t mode, struct block_device ** devptr)

Lock a device to prevent concurrent access.

Locking devices is necessary to prevent other programs or user from raw accessing the shadow device or the shield device. This is done by a call to `blkdev_get`.

`struct block_device **devptr` is a pointer to a memory region corresponding to a block device pointer. We will be update this region with the following code:

```
(*devptr)=dev
```

Definition at line 125 of file diffutil.c.

Referenced by `diffdev_try_open()`.

```

00126 {
00127     struct block_device* dev;
00128     if (!S_ISBLK(inode->i_mode))
00129         return -ENODEV;
00130     dev = bdget(kdev_t_to_nr(inode->i_rdev));
00131     if (!dev)
00132         return -ENOMEM;
00133     (*devptr)=dev;
00134     return blkdev_get(dev, mode, 0, BDEV_RAW);
00135 }

```

5.4.2.4 `int unlock_subdevice (struct block_device * dev)`

Unlock a device previously locked.

This function does the opposite of `lock_subdevice` except that it does not update any pointer. It simply unlock a block device.

Definition at line 142 of file `diffutil.c`.

Referenced by `diffdev_open()`.

```

00143 {
00144     if (!dev)
00145         return -ENODEV;
00146     return blkdev_put(dev, BDEV_RAW);
00147 }

```

5.4.2.5 `int compatible_devices (diffdev_device * dev)`

check compatibility of shield and shadow.

This function is checks the block sizes are the same on both devices an that shadow has a limited size (`blk_size[major]` is not null) which is non zero `blk_size[major][minor] >0`

Definition at line 154 of file `diffutil.c`.

Referenced by `allocate_header()`, and `init_device_from_scratch()`.

```

00155 {
00156     __u32 block_size1;
00157     __u32 block_size2;
00158     if (blksize_size[MAJOR(dev->kdev_shield)])
00159         block_size1=blksize_size[MAJOR(dev->kdev_shield)][MINOR(dev->kdev_shield)];
00160     else
00161         block_size1= BLOCK_SIZE;
00162     if (blksize_size[MAJOR(dev->kdev_shadow)])
00163         block_size2=blksize_size[MAJOR(dev->kdev_shadow)][MINOR(dev->kdev_shadow)];
00164     else
00165         block_size2= BLOCK_SIZE;
00166     if (block_size1 != block_size2)
00167         {

```

```

00168     printk ("soory\n %i <--> %i",block_size1, block_size2);
00169     return 0;
00170 }
00171 else
00172     return 1;
00173 }

```

5.4.2.6 diffdev_entry* allocate_header (diffdev_device * dev)

allocate a header.

This function allocate a header with needed size according to specified diffdev entry. returns NULL in case of error. the header is partially filled according to configuration. Most of prefilled fields are those who are not saved to disk

Definition at line 207 of file diffutil.c.

Referenced by load_device_from_disk().

```

00208 {
00209     __u32 header_size;
00210     __u32 superblock_count;
00211     __u32 superblock_size;
00212     __u32 block_size;
00213     __u32 dev1_blocks;
00214     __u32 i;
00215     diffdev_entry* device;
00216     if (!compatible_devices (dev))
00217     {
00218         printk ("<1>incompatible devices\n");
00219         return NULL;
00220     }
00221     printk ("<1>Hint: diffdev_table_entry is of size %d\n",sizeof (diffdev_table_entry));
00222     if (blksize_size[MAJOR(dev->kdev_shield)])
00223         block_size=blksize_size[MAJOR(dev->kdev_shield)][MINOR(dev->kdev_shield)];
00224     else
00225         block_size= BLOCK_SIZE;
00226     dev1_blocks=blk_size[MAJOR(dev->kdev_shadow)][MINOR(dev->kdev_shadow)]*1024/block_size;
00227     superblock_size=block_size/sizeof (diffdev_table_entry); /*how many shield can we put in one block*/
00228     superblock_count=(dev1_blocks/superblock_size)+1;
00229     /* disk mapped header datas is at least
00230     general datas+32bits*nr of superblocks.
00231     header size MUST be a multiple of block size */
00232     header_size= ( sizeof (diffdev_entry)
00233                 - offsetof (diffdev_entry,signature)
00234                 + superblock_count * sizeof (__u32)
00235                 );
00236     /* round to block_size*/
00237     header_size=(header_size/block_size)?((header_size/block_size)+1)*block_size:header_size;
00238     /*set device datas*/
00239     device=(void*)kmalloc (header_size+ offsetof (diffdev_entry,signature),GFP_KERNEL);
00240     SET_MAGIC (device,MAGIC_DIFFDEV_ENTRY);
00241     device->superblock=(diffdev_superblock_entry*)kmalloc (superblock_count*sizeof (diffdev_superblock_entry),GFP_KERNEL);
00242     device->superflags=(unsigned long*)kmalloc (sizeof (unsigned long)*superblock_count,GFP_KERNEL);
00243     device->header_size=header_size+ offsetof (diffdev_entry,signature);
00244     device->superblock_count=superblock_count; /*saved to disk but needed to check consistency*/
00245     device->superblock_size=superblock_size; /*saved to disk but needed to check consistency*/
00246     device->superblock_addr=(__u32*)((__u32)device+sizeof(diffdev_entry));

```

```

00247 device->nr_block_shadow=blk_size[MAJOR(dev->kdev_shadow)][MINOR(dev->kdev_shadow)]*1024/block_size;
00248 device->physical_datas=(void*)&device->signature;
00249 device->blksize=block_size;
00250 spin_lock_init (&device->lock);
00251 device->hardsect_size= get_defaultize(hardsect_size[MAJOR(dev->kdev_shadow)],
00252                                     MINOR(dev->kdev_shadow),
00253                                     512);
00254 if (blk_size[MAJOR(dev->kdev_shield)])
00255     device->nr_block_shield=blk_size[MAJOR(dev->kdev_shield)][MINOR(dev->kdev_shield)]*1024/block_size;
00256 else
00257     device->nr_block_shield=!((__u32)0);
00258 for (i=0;i<superblock_count;i++)
00259     {
00260     device->superblock[i].block=NULL;
00261     device->superflags[i]=0;/*clear all bits*/
00262     }
00263 return device;
00264 }

```

5.4.2.7 int load_device_from_disk (diffdev_device * dev)

create a diffdev header from disk.

When calling `init_device` with the `DIFFFLAG_ZEROSHLD` flag no set, we fall back in this function. It loads a header from the shield disk. The data structure is allocated from `allocate_header`.

Todo:

if the field `next_table` is set, load this one instead

Definition at line 288 of file `diffutil.c`.

Referenced by `init_device()`.

```

00289 {
00290     __u32 header_size;
00291     struct buffer_head* bh;
00292     diffdev_entry* device;
00293     device=allocate_header (dev);
00294     if (!device)
00295         return -ENOMEM;
00296     header_size=device->header_size-offsetof(diffdev_entry,signature);
00297     device->physical_address=0;
00298     printk (<1>getblk(dev->kdev_shield, %i, %i)\n",device->physical_address,header_size);
00299     bh = getblk(dev->kdev_shield, device->physical_address,header_size);
00301     if (!buffer_uptodate(bh))
00302     {
00303     //     /*Read if needed*/
00304     ll_rw_block(READ, 1, &bh);
00305     //     /*Wait for completion. Do not Suppose it worked good :(*/
00306     wait_on_buffer(bh);
00307     }
00308     memcpy (device->physical_datas,bh->b_data,header_size);
00309     printk (<1> memcpy (%p, %p, %d)\n",device->physical_datas,bh->b_data,header_size);
00310     dev->head=device;
00311     dump_device(device);
00312     brelse (bh);

```

```
00313 return 0;
00314 }
```

5.4.2.8 int init_device_from_scratch (diffdev_device * dev)

create a diffdev header from scratch.

When calling `init_device` with the `DIFFFLAG_ZEROSHLD` flag set, we fall back in this function. It creates a new header with all protection datas set so that every read is done from shadow device and every write is done by allocating a structure on shield. This simply means the shield is a new one.

All datas related to the shadow device (block_size a.s.o) are also calculated here.

Definition at line 327 of file `diffutil.c`.

Referenced by `init_device()`.

```
00328 {
00329     __u32 i,j;
00330     __u32 header_size;
00331     __u32 superbblock_count;
00332     __u32 superbblock_size;
00333     __u32 block_size;
00334     __u32 dev1_blocks;
00335     unsigned long flags;
00336     diffdev_entry* device;
00337     if (!compatible_devices (dev))
00338     {
00339         printk ("<1>incompatible devices\n");
00340         return -EINVAL;
00341     }
00342     /*
00343      * 2 Device initialisation
00344      */
00345     printk ("<1>Hint: diffdev_table_entry is of size %d\n",sizeof (diffdev_table_entry));
00346     if (blksize_size[MAJOR(dev->kdev_shield)])
00347         block_size=blksize_size[MAJOR(dev->kdev_shield)][MINOR(dev->kdev_shield)];
00348     else
00349         block_size= BLOCK_SIZE;
00350     dev1_blocks=blk_size[MAJOR(dev->kdev_shadow)][MINOR(dev->kdev_shadow)]*1024/block_size;
00351     superbblock_size=block_size/sizeof (diffdev_table_entry); /*how many shield can we put in one block*/
00352     superbblock_count=(dev1_blocks/superblock_size)+1;
00353     /* disk mapped header datas is at least
00354      general datas+32bits*nr of superblocks.
00355      header size MUST be a multiple of block size */
00356
00357     header_size= ( sizeof (diffdev_entry)
00358                   - offsetof (diffdev_entry,signature)
00359                   + superbblock_count * sizeof (__u32)
00360                 );
00361     /* round to block_size*/
00362     header_size=(header_size%block_size)?((header_size/block_size)+1)*block_size:header_size;
00363     /*Dump some infos*/
00364     printk ("<1>sizeof (diffdev_entry: %i\n",sizeof (diffdev_entry));
00365     printk ("<1>offsetof (diffdev_entry,superblock_size) %i\n",offsetof (diffdev_entry,superblock_size));
00366     printk ("<1>superblock_count * sizeof (__u32) %i\n",superblock_count * sizeof (__u32));
00367     /*set device datas*/
```

```

00368 device=(void*)kmalloc (header_size+ offsetof (diffdev_entry,signature),GFP_KERNEL);
00369 /* Since there is some space of entry not modified, set to 1 to prevent dumping
00370  * private datas to disk!
00371  */
00372 memset (device,1,header_size+ offsetof (diffdev_entry,signature));
00373 memcpy (device->id,".- Main root save point --.",32);
00374 memcpy (device->signature,"diffdev",7);
00375 memcpy (device->version,"0.0.01",7);
00376 device->header_size=header_size+ offsetof (diffdev_entry,signature);
00377 device->superblock_count=superblock_count;
00378 device->superblock_size=superblock_size;
00379 device->superblock_addr=(__u32*)((__u32)device+sizeof(diffdev_entry));
00380 device->current_free_block=0;
00381 device->superblock=(diffdev_superblock_entry*)kmalloc (sizeof (diffdev_superblock_entry)*device->superb
00382 device->superflags=(unsigned long*)kmalloc (sizeof (unsigned long)*superblock_count,GFP_KERNEL);
00383 device->physical_datas=(void*)&device->signature);
00384 device->nr_block_shadow=blk_size[MAJOR(dev->kdev_shadow)] [MINOR(dev->kdev_shadow)]*1024/block_size;
00385 SET_MAGIC (device,MAGIC_DIFFDEV_ENTRY);
00386 if (blk_size[MAJOR(dev->kdev_shield)])
00387     device->nr_block_shield=blk_size[MAJOR(dev->kdev_shield)] [MINOR(dev->kdev_shield)]*1024/block_size;
00388 else
00389     device->nr_block_shield=!((__u32)0);
00390 device->blksize=block_size;
00391 device->hardsect_size= get_defaultize(hardsect_size[MAJOR(dev->kdev_shadow)],
00392                                     MINOR(dev->kdev_shadow),
00393                                     512);
00394 spin_lock_init (&device->lock);
00395 /*At startup, no superblock exists yet and table start located at 1st block*/
00396 for (i=0;i<device->superblock_count;i++)
00397     {
00398     device->superblock_addr[i]=0;
00399     device->superflags[i]=0;
00400     device->superblock[i].block=NULL;
00401     }
00402 device->physical_address=0;
00403
00404 header_size=sizeof(__u32)*(device->superblock_count)
00405             +sizeof (diffdev_entry)
00406             -offsetof (diffdev_entry,signature);
00407 printk ("device.superblock_count is %d , header_size is %d , device entry is %d bytes long\n",
00408         device->superblock_count,header_size, header_size+ offsetof (diffdev_entry,signature));
00409 printk ("device structure ranges from %u to %u (%u+%u)\n",
00410         device,
00411         ((__u32)device+((__u32)header_size+ (__u32)offsetof(diffdev_entry,signature))),
00412         device, header_size, offsetof(diffdev_entry,signature));
00413 j=0;
00414 /*Let's allocate blocks needed at start of device.
00415  We do this by calling several time get_free_block*/
00416 for (;header_size>block_size;header_size-=block_size)
00417     get_free_block(device,LOCK_NONE);
00418 get_free_block(device,LOCK_NONE);
00419 /*let's acquire the lock...*/
00420 write_lock_irqsave (&dev->rwlock_config,flags);
00421 dev->head=device;
00422 write_unlock_irqrestore (&dev->rwlock_config,flags);
00423 save_header (dev);
00424 return 0;
00425 }

```

5.4.2.9 int init_device (diffdev_device * dev)

Init a diffdev device.

When the shadow and shield devices are correctly locked, init_device is called to load the header of the diffdev, check it's sanity and eventually, do any transaction operation awaiting.

Definition at line 435 of file diffutil.c.

Referenced by diffdev_open().

```

00436 {
00437     int result;
00438     if TEST_FLAG (dev,DIFFFLAG_ZEROSHLD){
00439         CLEAR_FLAG (dev,DIFFFLAG_ZEROSHLD);
00440         printk ("<1>creating from scratch\n");
00441         if ((result= init_device_from_scratch (dev)))
00442             return result;
00443     }
00444     else{
00445         if ((result=load_device_from_disk (dev)))
00446             return result;
00447     }
00448     return 0;
00449 }

```

6 differential block device Page Documentation

6.1 Todo List

Global cleanup_module(void) FIXME: support for max_readahead and max_sectors

Global init_module(void) FIXME: handling max_sectors according to underlying structure

Global DIFFDEV_SIZE FIXME size depend on lower disk size.

Global get_free_block(diffdev_entry (p.3) *dev, int lockmode) When running out of free block, wakeup any awaiting daemon.
Really out of free block: go to emergency read only

Global load_device_from_disk(diffdev_device (p.2) *dev) if the field next_table is set, load this one instead

Index

- ACQUIRE_CONFIG
 - diffmod.c, 14
 - allocate_header
 - diffutil.c, 46
 - atome_config
 - diffmod.c, 14
 - bdev_shadow
 - diffdev_device, 2
 - bdev_shield
 - diffdev_device, 2
 - blksize
 - diffdev_entry, 4
 - block
 - diffdev_superblock_entry, 6
 - block2sector
 - difftransfert.c, 33
 - block_span
 - diffdev_entry, 4
 - check_diffdev_number
 - diffmod.c, 12
 - cleanup_module
 - diffmod.c, 25
 - CLEAR_FLAG
 - diffmod.h, 28
 - compatible_devices
 - diffutil.c, 45
 - CONFIG_READ_LOCK
 - diffmod.c, 14
 - CONFIG_READ_UNLOCK
 - diffmod.c, 14
 - CONFIG_WRITE_LOCK
 - diffmod.c, 14
 - CONFIG_WRITE_UNLOCK
 - diffmod.c, 15
 - COR
 - diffdev_table_entry, 7
 - current_free_block
 - diffdev_entry, 5
 - DEBUG
 - diffmod.c, 11
 - difftransfert.c, 32
 - device
 - sub_device_op, 8
 - superblock_op, 8
 - superblock_w_op, 9
 - DEVICE_INTR
 - diffmod.c, 11
 - DEVICE_NAME
 - diffmod.c, 11
 - device_nbr
 - diffdev_params, 6
 - DEVICE_NO_RANDOM
 - diffmod.c, 11
 - DEVICE_NR
 - diffmod.c, 11
 - DEVICE_REQUEST
 - diffmod.c, 11
 - diffdev_bdops
 - diffmod.c, 26
 - DIFFDEV_BLKSIZE
 - diffmod.h, 28
 - diffdev_blksize
 - diffmod.c, 13
 - diffdev_blksizes
 - diffmod.c, 13
 - diffdev_checkchange
 - diffmod.c, 22
 - diffdev_device, 2
 - bdev_shadow, 2
 - bdev_shield, 2
 - flags, 2
 - head, 2
 - kdev_shadow, 2
 - kdev_shield, 2
 - open, 2
 - rwlock_config, 3
 - sem_config, 3
 - shadowname, 3
 - shieldname, 3
 - valid, 2
 - DIFFDEV_DEVS
 - diffmod.h, 28
 - diffdev_devs
 - diffmod.c, 13
 - DIFFDEV_EMERGENCY
 - diffmod.h, 27
 - diffdev_entry, 3
 - blksize, 4
 - block_span, 4
 - current_free_block, 5
 - hardsect_size, 4
-

- header_size, 4
- id, 5
- lock, 4
- magic, 5
- needsave, 4
- next_table, 6
- nr_block_shadow, 4
- nr_block_shield, 4
- physical_address, 4
- physical_datas, 4
- signature, 4
- superblock, 4
- superblock_addr, 5
- superblock_count, 5
- superblock_size, 5
- superflags, 5
- version, 4
- DIFFDEV_HARDSECT
 - diffmod.h, 28
- diffdev_hardsect
 - diffmod.c, 13
- diffdev_hardsects
 - diffmod.c, 13
- DIFFDEV_IO_GETPARAMS
 - diffmod.h, 27
- DIFFDEV_IO_SETPARAMS
 - diffmod.h, 27
- DIFFDEV_IO_ZEROSHIELD
 - diffmod.h, 30
- diffdev_ioctl
 - diffmod.c, 22
- DIFFDEV_IOCTL_BASE
 - diffmod.h, 27
- diffdev_ioctl_params
 - diffmod.c, 20
- diffdev_is_open
 - diffmod.c, 12
- diffdev_lock
 - diffmod.c, 13
- DIFFDEV_MAILTO
 - diffmod.h, 28
- DIFFDEV_MAJOR
 - diffmod.h, 28
- diffdev_major
 - diffmod.c, 13
- diffdev_make_request
 - diffmod.c, 15
- diffdev_open
 - diffmod.c, 16
- diffdev_params, 6
 - device_nbr, 6
 - shadowname, 6
 - shieldname, 6
- DIFFDEV_RAHEAD
 - diffmod.h, 28
- diffdev_rahead
 - diffmod.c, 13
- diffdev_release
 - diffmod.c, 18
- diffdev_revalidate
 - diffmod.c, 23
- DIFFDEV_SIZE
 - diffmod.h, 30
- diffdev_size
 - diffmod.c, 13
- diffdev_sizes
 - diffmod.c, 13
- diffdev_superblock_entry, 6
 - block, 6
- diffdev_table_entry, 7
 - COR, 7
 - MOD, 7
 - shield, 7
- diffdev_try_open
 - diffutil.c, 43
- diffdevs
 - diffmod.c, 26
- DIFFFLAG_NONE
 - diffmod.h, 28
- DIFFFLAG_NOPE
 - diffmod.h, 28
- DIFFFLAG_ZEROSHLD
 - diffmod.h, 28
- DIFFINFO
 - diffmod.h, 28
- diffmod.c, 10
 - ACQUIRE_CONFIG, 14
 - atome_config, 14
 - check_diffdev_number, 12
 - cleanup_module, 25
 - CONFIG_READ_LOCK, 14
 - CONFIG_READ_UNLOCK, 14
 - CONFIG_WRITE_LOCK, 14
 - CONFIG_WRITE_UNLOCK, 15
 - DEBUG, 11
 - DEVICE_INTR, 11
 - DEVICE_NAME, 11
 - DEVICE_NO_RANDOM, 11

- DEVICE_NR, 11
- DEVICE_REQUEST, 11
- diffdev_bdops, 26
- diffdev_blksize, 13
- diffdev_blksizes, 13
- diffdev_checkchange, 22
- diffdev_devs, 13
- diffdev_hardsect, 13
- diffdev_hardsects, 13
- diffdev_ioctl, 22
- diffdev_ioctl_params, 20
- diffdev_is_open, 12
- diffdev_lock, 13
- diffdev_major, 13
- diffdev_make_request, 15
- diffdev_open, 16
- diffdev_rahead, 13
- diffdev_release, 18
- diffdev_revalidate, 23
- diffdev_size, 13
- diffdev_sizes, 13
- diffdevs, 26
- init_module, 23
- is_config_device, 12
- MAJOR_NR, 11
- major_shadow, 13
- major_shield, 13
- minor_shadow, 13
- minor_shield, 13
- MODULE, 11
- MODULE_AUTHOR, 12
- MODULE_DESCRIPTION, 12
- MODULE_LICENSE, 12
- MODULE_PARM, 12
- MODULE_PARM_DESC, 12
- RELEASE_CONFIG, 11
- shadow, 13
- shield, 14
- valid_parameters, 13
- diffmod.h, 26
 - CLEAR_FLAG, 28
 - DIFFDEV_BLKSIZE, 28
 - DIFFDEV_DEVS, 28
 - DIFFDEV_EMERGENCY, 27
 - DIFFDEV_HARDSECT, 28
 - DIFFDEV_IO_GETPARAMS, 27
 - DIFFDEV_IO_SETPARAMS, 27
 - DIFFDEV_IO_ZEROSHIELD, 30
 - DIFFDEV_IOCTL_BASE, 27
 - DIFFDEV_MAILTO, 28
 - DIFFDEV_MAJOR, 28
 - DIFFDEV_RAHEAD, 28
 - DIFFDEV_SIZE, 30
 - DIFFFLAG_NONE, 28
 - DIFFFLAG_NOPE, 28
 - DIFFFLAG_ZEROSHLD, 28
 - DIFFINFO, 28
 - get_defaultize, 28
 - LOCK_IRQ, 29
 - LOCK_NONE, 29
 - LOCK_SIMPLE, 29
 - MAGIC_DIFFDEV_BASE, 30
 - MAGIC_DIFFDEV_ENTRY, 31
 - MAGIC_DIFFDEV_SBLKOP, 31
 - MAGIC_DIFFDEV_SBLKW, 31
 - MAGIC_DIFFDEV_SUBOP, 31
 - MAX_DEVICE_NAME_LEN, 27
 - NEED_MAGIC, 30
 - offsetof, 30
 - SB_Loading, 29
 - SB_NeedSave, 29
 - SB_opswaiting, 29
 - SB_Saving, 29
 - SET_FLAG, 28
 - SET_MAGIC, 29
 - superblock_op, 31
 - TEST_FLAG, 28
- difftransfert.c, 31
 - block2sector, 33
 - DEBUG, 32
 - end_block_operation, 34
 - end_stub_operation, 35
 - end_sub_operation, 35
 - finnish_superblock, 38
 - MODULE, 32
 - prepare_superblock, 37
 - read_superblock, 36
 - rw_block_physical, 39
 - rw_block_shadow, 40
 - rw_block_shield, 39
 - sector2block, 33

- try_read_block, 40
- try_write_block, 33
- diffutil.c, 41
 - allocate_header, 46
 - compatible_devices, 45
 - diffdev_try_open, 43
 - dump_device, 42
 - get_free_block, 43
 - init_device, 49
 - init_device_from_scratch, 48
 - is_valid_request, 42
 - load_device_from_disk, 47
 - lock_subdevice, 44
 - MODULE, 42
 - save_header, 42
 - unlock_subdevice, 45
- dump_device
 - diffutil.c, 42
- end_block_operation
 - difftransfert.c, 34
- end_stub_operation
 - difftransfert.c, 35
- end_sub_operation
 - difftransfert.c, 35
- finnish_superblock
 - difftransfert.c, 38
- flags
 - diffdev_device, 2
- get_defaultize
 - diffmod.h, 28
- get_free_block
 - diffutil.c, 43
- hardsect_size
 - diffdev_entry, 4
- head
 - diffdev_device, 2
- header_size
 - diffdev_entry, 4
- id
 - diffdev_entry, 5
- init_device
 - diffutil.c, 49
- init_device_from_scratch
 - diffutil.c, 48
- init_module
 - diffmod.c, 23
- is_config_device
 - diffmod.c, 12
- is_valid_request
 - diffutil.c, 42
- kdev_shadow
 - diffdev_device, 2
- kdev_shield
 - diffdev_device, 2
- load_device_from_disk
 - diffutil.c, 47
- lock
 - diffdev_entry, 4
- LOCK_IRQ
 - diffmod.h, 29
- LOCK_NONE
 - diffmod.h, 29
- LOCK_SIMPLE
 - diffmod.h, 29
- lock_subdevice
 - diffutil.c, 44
- magic
 - diffdev_entry, 5
 - sub_device_op, 8
 - superblock_op, 9
 - superblock_w_op, 10
- MAGIC_DIFFDEV_BASE
 - diffmod.h, 30
- MAGIC_DIFFDEV_ENTRY
 - diffmod.h, 31
- MAGIC_DIFFDEV_SBLKOP
 - diffmod.h, 31
- MAGIC_DIFFDEV_SBLKW
 - diffmod.h, 31
- MAGIC_DIFFDEV_SUBOP
 - diffmod.h, 31
- MAJOR_NR
 - diffmod.c, 11
- major_shadow
 - diffmod.c, 13
- major_shield
 - diffmod.c, 13
- MAX_DEVICE_NAME_LEN
 - diffmod.h, 27
- minor_shadow
 - diffmod.c, 13
- minor_shield
 - diffmod.c, 13

- MOD
 - diffdev_table_entry, 7
- MODULE
 - diffmod.c, 11
 - difftransfert.c, 32
 - diffutil.c, 42
- MODULE_AUTHOR
 - diffmod.c, 12
- MODULE_DESCRIPTION
 - diffmod.c, 12
- MODULE_LICENSE
 - diffmod.c, 12
- MODULE_PARM
 - diffmod.c, 12
- MODULE_PARM_DESC
 - diffmod.c, 12
- NEED_MAGIC
 - diffmod.h, 30
- needsave
 - diffdev_entry, 4
- next
 - superblock_op, 9
- next_table
 - diffdev_entry, 6
- nr_block_shadow
 - diffdev_entry, 4
- nr_block_shield
 - diffdev_entry, 4
- offsetof
 - diffmod.h, 30
- op_type
 - sub_device_op, 8
 - superblock_op, 9
- open
 - diffdev_device, 2
- physical_address
 - diffdev_entry, 4
- physical_datas
 - diffdev_entry, 4
- prepare_superblock
 - difftransfert.c, 37
- read_superblock
 - difftransfert.c, 36
- RELEASE_CONFIG
 - diffmod.c, 11
- request
 - sub_device_op, 7
 - superblock_op, 8
 - rw_block_physical
 - difftransfert.c, 39
 - rw_block_shadow
 - difftransfert.c, 40
 - rw_block_shield
 - difftransfert.c, 39
 - rwlock_config
 - diffdev_device, 3
 - save_header
 - diffutil.c, 42
 - SB.Loading
 - diffmod.h, 29
 - SB.NeedSave
 - diffmod.h, 29
 - SB.opswaiting
 - diffmod.h, 29
 - SB.Saving
 - diffmod.h, 29
 - sector2block
 - difftransfert.c, 33
 - sem_config
 - diffdev_device, 3
 - SET_FLAG
 - diffmod.h, 28
 - SET_MAGIC
 - diffmod.h, 29
 - shadow
 - diffmod.c, 13
 - shadowname
 - diffdev_device, 3
 - diffdev_params, 6
 - shield
 - diffdev_table_entry, 7
 - diffmod.c, 14
 - shieldname
 - diffdev_device, 3
 - diffdev_params, 6
 - signature
 - diffdev_entry, 4
 - sub_device_op, 7
 - device, 8
 - magic, 8
 - op_type, 8
 - request, 7
 - try_count, 8
 - superblock
 - diffdev_entry, 4

- superblock_addr
 - diffdev_entry, 5
- superblock_count
 - diffdev_entry, 5
- superblock_nr
 - superblock_op, 9
 - superblock_w_op, 10
- superblock_op, 8
 - device, 8
 - diffmod.h, 31
 - magic, 9
 - next, 9
 - op_type, 9
 - request, 8
 - superblock_nr, 9
 - try_count, 9
- superblock_size
 - diffdev_entry, 5
- superblock_w_op, 9
 - device, 9
 - magic, 10
 - superblock_nr, 10
- superflags
 - diffdev_entry, 5

- TEST_FLAG
 - diffmod.h, 28
- try_count
 - sub_device_op, 8
 - superblock_op, 9
- try_read_block
 - difftransfert.c, 40
- try_write_block
 - difftransfert.c, 33

- unlock_subdevice
 - diffutil.c, 45

- valid
 - diffdev_device, 2
- valid_parameters
 - diffmod.c, 13
- version
 - diffdev_entry, 4