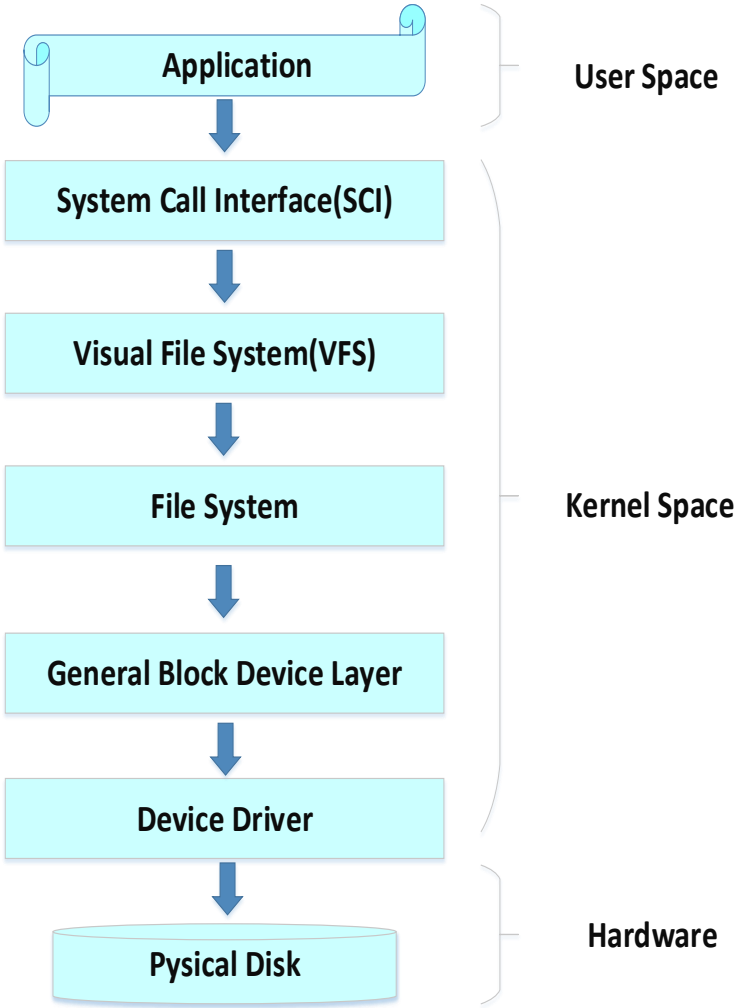
EXT2 File System

Youxu Chen

USTC, CHINA
ADSLAB
先进数据系统实验室

Overview

- File System
  - VFS
  - Specified FS
    - EXT family
    - BTRFS



VFS - Virtual File System

- What is VFS?
  - an abstraction layer on top of a more concrete file system
- Actions
  - Manages kernel level file abstractions in one format for all file systems
  - Receives system call requests from user level (e.g. write, open, stat, link)
  - Interacts with a specific file system based on mount point traversal

EXT2 File System

- Introduction
  - The extended file system family
    - EXT
    - EXT2:the second extended file system
    - EXT3 and EXT4
      - EXT2+Journal
- Why is EXT2?
  - Fundamental data structure
  - Simple

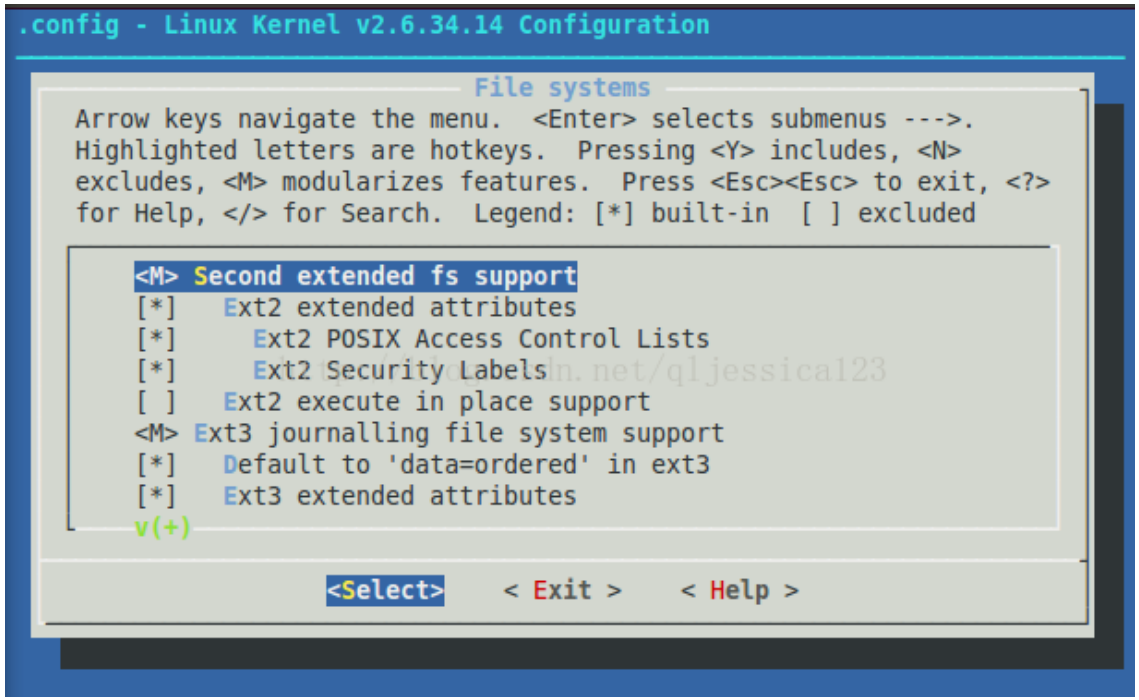
Source Code

- Download
  - https://www.kernel.org/
- EXT2 directory
  - Code
  - Makefile

acl.c	2015/5/7 4:04	C 文件	6 KB
acl.h	2015/5/7 4:04	H 文件	2 KB
balloc.c	2015/5/7 4:04	C 文件	45 KB
dir.c	2015/5/7 4:04	C 文件	18 KB
ext2.h	2015/5/7 4:04	H 文件	28 KB
file.c	2015/5/7 4:04	C 文件	4 KB
ialloc.c	2015/5/7 4:04	C 文件	19 KB
inode.c	2015/5/7 4:04	C 文件	45 KB
ioctl.c	2015/5/7 4:04	C 文件	5 KB
Kconfig	2015/5/7 4:04	文件	2 KB
Makefile	2015/5/7 4:04	文件	1 KB
namei.c	2015/5/7 4:04	C 文件	10 KB
super.c	2015/5/7 4:04	C 文件	43 KB
symlink.c	2015/5/7 4:04	C 文件	2 KB
xattr.c	2015/5/7 4:04	C 文件	28 KB
xattr.h	2015/5/7 4:04	H 文件	4 KB
xattr_security.c	2015/5/7 4:04	C 文件	2 KB
xattr_trusted.c	2015/5/7 4:04	C 文件	2 KB
xattr_user.c	2015/5/7 4:04	C 文件	2 KB

Make Module

- Compile Linux kernel
  - `make menuconfig`
  - `make`
  - `make modules_install`
  - `make install`

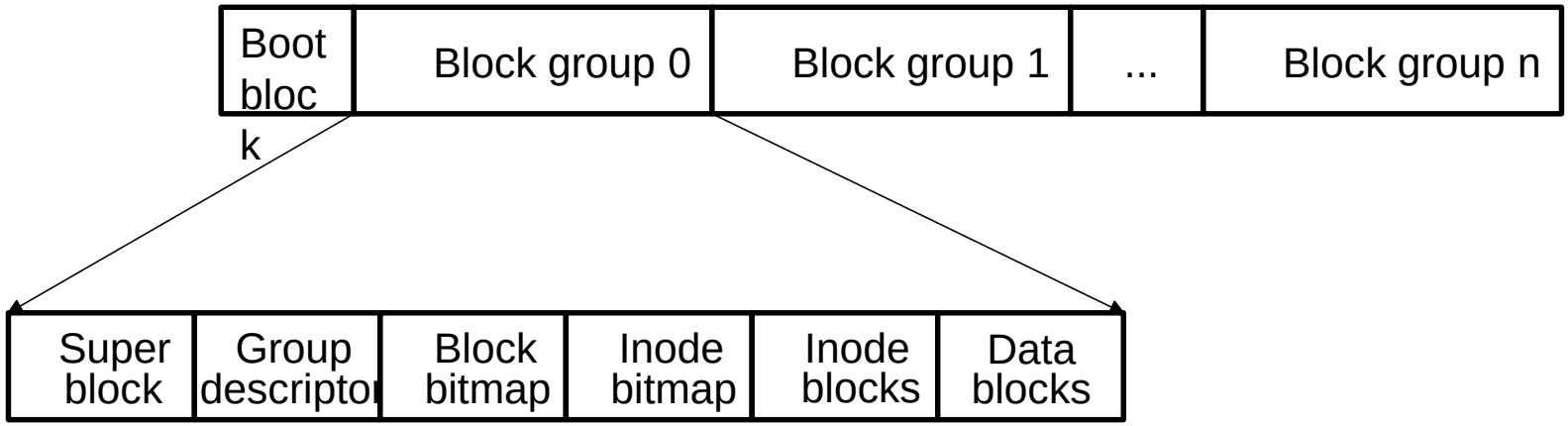


Make EXT2 Module

- Makefile
  - http://blog.csdn.net/ruglcc/article/details/7814546/
- New makefile
  - http://blog.csdn.net/chenyouxu/article/details/46985909
  - Module
    - ext2.ko
  - Insert module
    - `insmod ext2.ko`

Now, it is yours.

Disk Layout



- Block group(defined in ext2.h)
  - Super block
  - Group descriptor
  - Block bitmap
  - Inode bitmap
  - Inode blocks
  - data blocks

Super Block

- Core structure of file system metadata
- Info of whole FS
  - Inodes count
  - Blocks count
  - Block size
  - Inodes per group etc.

```
/*
 * Structure of the super block
 */
struct ext2_super_block {
    __le32  s_inodes_count;     /* Inodes count */
    __le32  s_blocks_count;     /* Blocks count */
    __le32  s_r_blocks_count;   /* Reserved blocks count */
    __le32  s_free_blocks_count;    /* Free blocks count */
    __le32  s_free_inodes_count;    /* Free inodes count */
    __le32  s_first_data_block; /* First Data Block */
    __le32  s_log_block_size;   /* Block size */
    __le32  s_log_frag_size;    /* Fragment size */
    __le32  s_blocks_per_group; /* # Blocks per group */
    __le32  s_frags_per_group;  /* # Fragments per group */
    __le32  s_inodes_per_group; /* # Inodes per group */
    __le32  s_mtime;        /* Mount time */
    __le32  s_wtime;        /* Write time */
    __le16  s_mnt_count;        /* Mount count */
    __le16  s_max_mnt_count;    /* Maximal mount count */
    __le16  s_magic;        /* Magic signature */
    __le16  s_state;        /* File system state */
    __le16  s_errors;       /* Behaviour when detecting errors */
    __le16  s_minor_rev_level;  /* minor revision level */
    __le32  s_lastcheck;        /* time of last check */
    __le32  s_checkinterval;    /* max. time between checks */
    __le32  s_creator_os;       /* OS */
    __le32  s_rev_level;        /* Revision level */
    __le16  s_def_resuid;       /* Default uid for reserved blocks */
    __le16  s_def_resgid;       /* Default gid for reserved blocks */
```

Group Descriptor

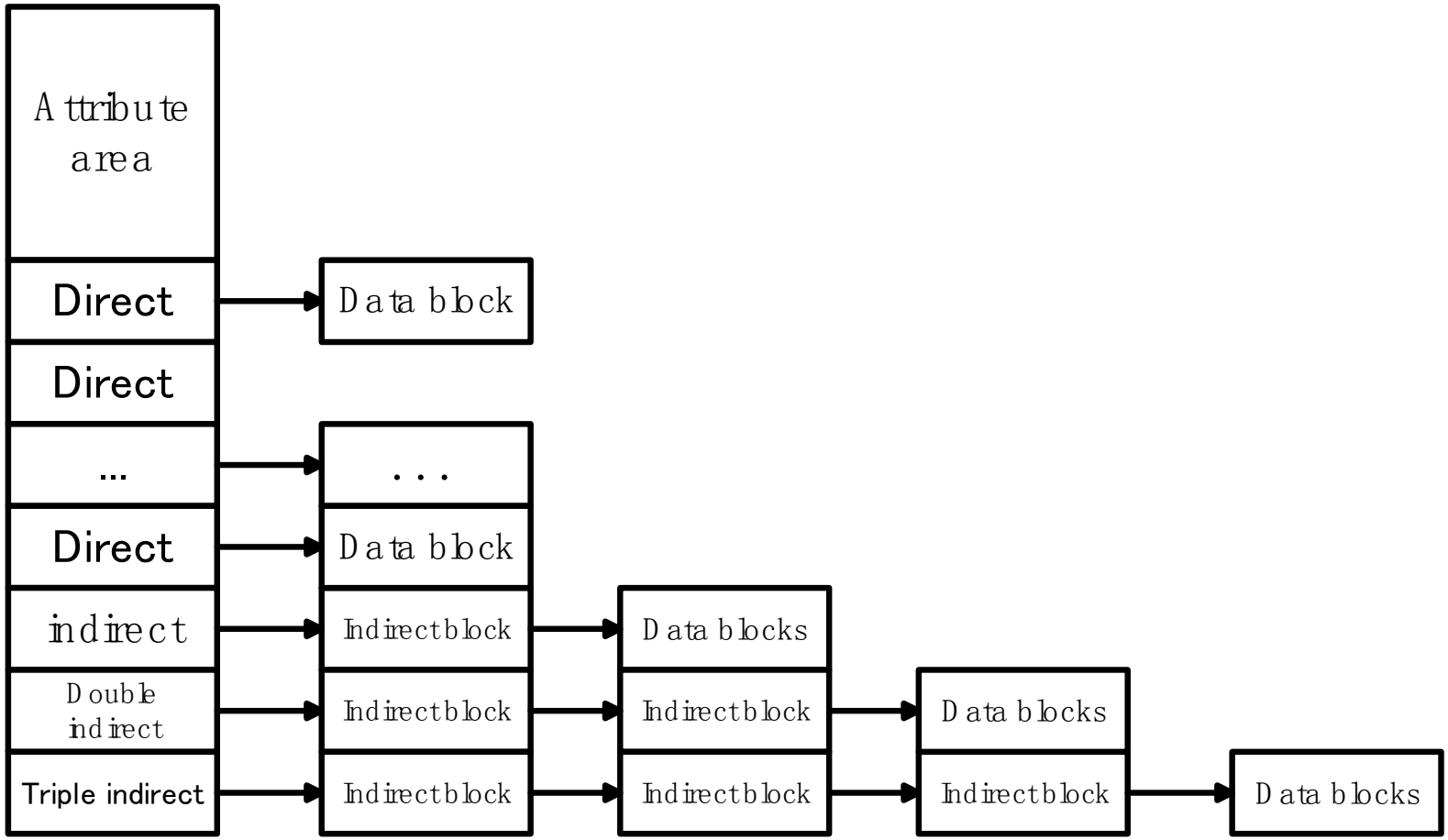
- Metadata of block group

```
struct ext2_group_desc
{
    __le32  bg_block_bitmap;        /* Blocks bitmap block */
    __le32  bg_inode_bitmap;        /* Inodes bitmap block */
    __le32  bg_inode_table;     /* Inodes table block */
    __le16  bg_free_blocks_count;   /* Free blocks count */
    __le16  bg_free_inodes_count;   /* Free inodes count */
    __le16  bg_used_dirs_count; /* Directories count */
    __le16  bg_pad;
    __le32  bg_reserved[3];
};
```

Bitmap

- A long bit
  - One bit stands for the use status of one block in this group
- Block bitmap
- Inode bitmap

Inode Structure



Inode

- Metadata of one file/directory

```
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
    __le16  i_mode;     /* File mode */
    __le16  i_uid;      /* Low 16 bits of Owner Uid */
    __le32  i_size;     /* Size in bytes */
    __le32  i_atime;    /* Access time */
    __le32  i_ctime;    /* Creation time */
    __le32  i_mtime;    /* Modification time */
    __le32  i_dtime;    /* Deletion Time */
    __le16  i_gid;      /* Low 16 bits of Group Id */
    __le16  i_links_count;  /* Links count */
    __le32  i_blocks;   /* Blocks count */
    __le32  i_flags;    /* File flags */
    union {
        struct {
            __le32  l_i_reserved1;
        } linux1;
        struct {
            __le32  h_i_translator;
        } hurd1;
        struct {
            __le32  m_i_reserved1;
        } masix1;
    } osd1;             /* OS dependent 1 */
    __le32  i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
    __le32  i_generation;   /* File version (for NFS) */
    __le32  i_file_acl; /* File ACL */
```

Directory

- Contains multiple files or sub-directories
- How to find a file in this directory?
  - Directory entry

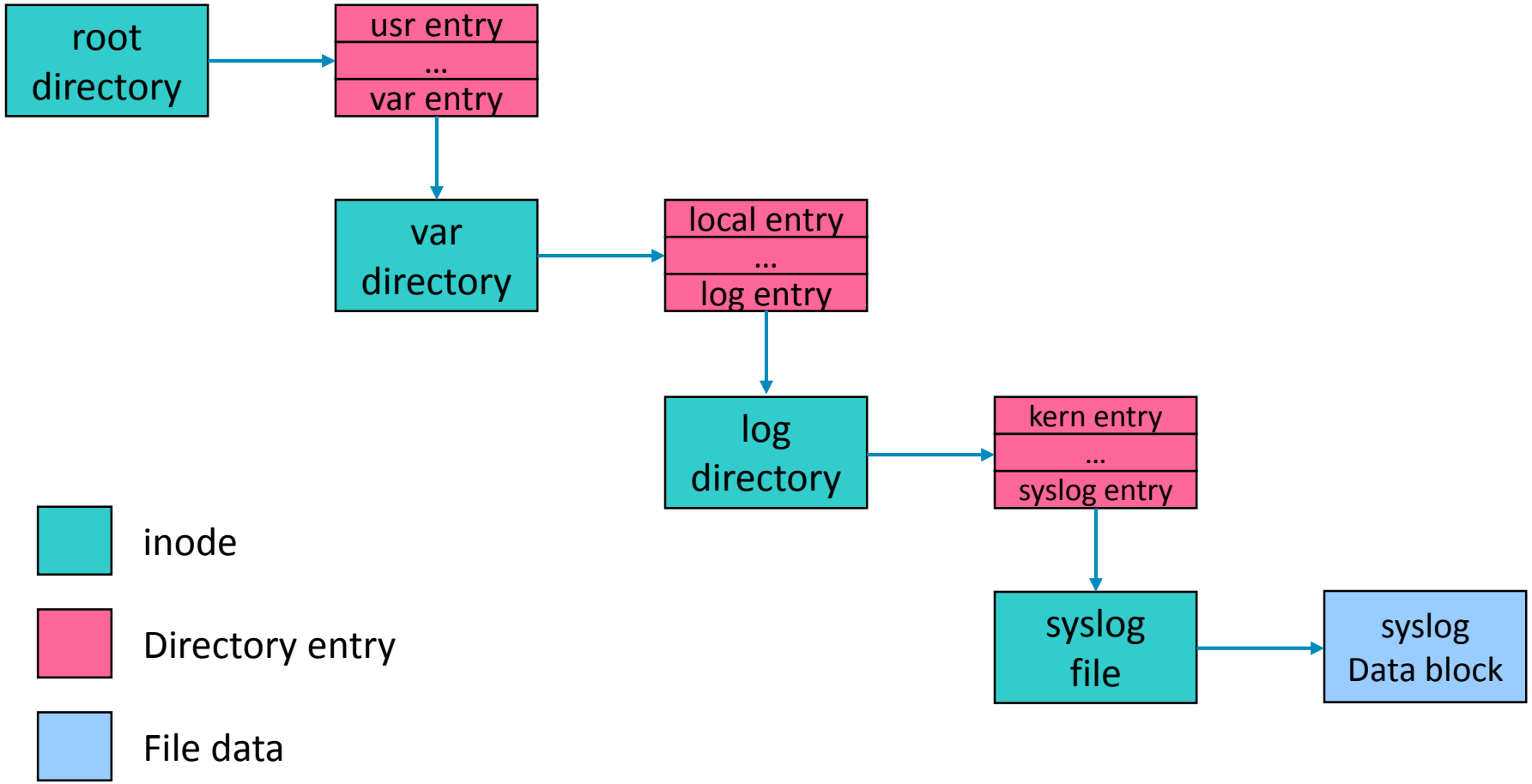
```
struct ext2_dir_entry_2 {
    __le32  inode;          /* Inode number */
    __le16  rec_len;        /* Directory entry length */
    __u8    name_len;       /* Name length */
    __u8    file_type;
    char    name[];         /* File name, up to EXT2_NAME_LEN */
};
```

25	14	6	‘f’	File_A
26	14	6	‘f’	File_B
28	13	5	‘D’	Dir_A

Directory entry

Data block of directory

Cat /var/log/syslog



Note

- VFS vs. EXT2
  - inode vs. ext2_inode

```
struct inode {
    umode_t         i_mode;
    unsigned short      i_opflags;
    kuid_t          i_uid;
    kgid_t          i_gid;
    unsigned int        i_flags;

#ifdef CONFIG_FS_POSIX_ACL
    struct posix_acl    *i_acl;
    struct posix_acl    *i_default_acl;
#endif

    const struct inode_operations   *i_op;
    struct super_block  *i_sb;
    struct address_space    *i_mapping;
    const struct file_operations    *i_fop;
```

```
struct ext2_inode {
    __le16  i_mode;     /* File mode */
    __le16  i_uid;      /* Low 16 bits of Owner Uid */
    __le32  i_size;     /* Size in bytes */
    __le32  i_atime;    /* Access time */
    __le32  i_ctime;    /* Creation time */
    __le32  i_mtime;    /* Modification time */
    __le32  i_dtime;    /* Deletion Time */
    __le16  i_gid;      /* Low 16 bits of Group Id */
    __le16  i_links_count;  /* Links count */
    __le32  i_blocks;   /* Blocks count */
    __le32  i_flags;    /* File flags */
```

  - super_block vs. ext2_super_block
- Data structure of VFS is defined in <include/linux/fs.h>

File & inode Operations

- Defined in file.c
- File operations
  - Open
  - Read
  - Write

```
const struct file_operations ext2_file_operations = {
    .llseek      = generic_file_llseek,
    .read_iter   = ext2_file_read_iter,
    .write_iter  = ext2_file_write_iter,
    .unlocked_ioctl = ext2_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl    = ext2_compat_ioctl,
#endif
    .mmap        = ext2_file_mmap,
    .open        = dquot_file_open,
    .release     = ext2_release_file,
    .fsync       = ext2_fsync,
    .get_unmapped_area = thp_get_unmapped_area,
    .splice_read     = generic_file_splice_read,
    .splice_write    = iter_file_splice_write,
};
```

- Inode operations
  - Setattr
  - Set_acl
  - Get_acl

```
const struct inode_operations ext2_file_inode_operations = {
#ifdef CONFIG_EXT2_FS_XATTR
    .listxattr   = ext2_listxattr,
#endif
    .setattr     = ext2_setattr,
    .get_acl     = ext2_get_acl,
    .set_acl     = ext2_set_acl,
    .fiemap      = ext2_fiemap,
};
```

Open

- Given a pathname for a file, open() returns a file descriptor, a small, nonnegative integer for use in subsequent system calls
  - `int open(const char *pathname, int flags, mode_t mode)`
- Flags
  - Access modes
    - O_RDONLY, O_WRONLY, O_RDWR
  - File creation flags
    - O_CREAT, O_DIRECORY
- Mode

S_IRWXU	00700	user (file owner) has read, write, and execute permission
S_IRUSR	00400	user has read permission
S_IWUSR	00200	user has write permission
S_IXUSR	00100	user has execute permission
S_IRWXG	00070	group has read, write, and execute permission
S_IRGRP	00040	group has read permission
S_IWGRP	00020	group has write permission
S_IXGRP	00010	group has execute permission
S_IRWXO	00007	others have read, write, and execute permission
S_IROTH	00004	others have read permission
S_IWOTH	00002	others have write permission
S_IXOTH	00001	others have execute permission

Open

- Process – file

```
struct task_struct{
..
/*文件系统信息*/
struct fs_struct *fs;
...
/*打开文件信息*/
struct files_struct *files;
...
}
```

```
struct files_struct{
atomic_t count;
struct fdtable *fdt;
struct fdtable fdtab;
int next_fd;
...
}
```

```
struct file {
struct path f_path;
...
}
```

```
struct path{
struct vfsmount *mnt;
struct dentry *dentry;
}
```

- File descriptor is the index of fdtable array

Open

- System call
  - <fs/open.c>
  - A smaple
    - Open file with O_CREAT

```
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
    if (force_o_largefile())
        flags |= O_LARGEFILE;

    return do_sys_open(AT_FDCWD, filename, flags, mode);
}
```

do_sys_open

```
long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
{
    struct open_flags op;
    int fd = build_open_flags(flags, mode, &op);
    struct filename *tmp;

    if (fd)
        return fd;

    tmp = getname(filename);
    if (IS_ERR(tmp))
        return PTR_ERR(tmp);

    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
        struct file *f = do_filp_open(dfd, tmp, &op);
        if (IS_ERR(f)) {
            put_unused_fd(fd);
            fd = PTR_ERR(f);
        } else {
            fsnotify_open(f);
            fd_install(fd, f);
        }
    }
    putname(tmp);
    return fd;
}
```

do_filp_open

```
struct file *do_filp_open(int dfd, struct filename *pathname,
        const struct open_flags *op)
{
    struct nameidata nd;
    int flags = op->lookup_flags;
    struct file *filp;

    set_nameidata(&nd, dfd, pathname);
    filp = path_openat(&nd, op, flags | LOOKUP_RCU);
    if (unlikely(filp == ERR_PTR(-ECHILD)))
        filp = path_openat(&nd, op, flags);
    if (unlikely(filp == ERR_PTR(-ESTALE)))
        filp = path_openat(&nd, op, flags | LOOKUP_REVAL);
    restore_nameidata();
    return filp;
}
```

path_openat

```
static struct file *path_openat(struct nameidata *nd,
            const struct open_flags *op, unsigned flags)
{
    const char *s;
    struct file *file;
    int opened = 0;
    int error;

    file = get_empty_filp();
    if (IS_ERR(file))
        return file;

    file->f_flags = op->open_flag;

    if (unlikely(file->f_flags & __O_TMPFILE)) {
        error = do_tmpfile(nd, flags, op, file, &opened);
        goto out2;
    }

    if (unlikely(file->f_flags & O_PATH)) {
        error = do_o_path(nd, flags, file);
        if (!error)
            opened |= FILE_OPENED;
        goto out2;
    }

    s = path_init(nd, flags);
    if (IS_ERR(s)) {
        put_filp(file);
        return ERR_CAST(s);
    }
    while (!(error = link_path_walk(s, nd)) &&
        (error = do_last(nd, file, op, &opened)) > 0) {
        nd->flags &= ~(LOOKUP_OPEN|LOOKUP_CREATE|LOOKUP_EXCL);
        s = trailing_symlink(nd);
        if (IS_ERR(s)) {
            error = PTR_ERR(s);
            break;
```

do_last

```
/*
 * Handle the last step of open()
 */
static int do_last(struct nameidata *nd,
            struct file *file, const struct open_flags *op,
            int *opened)
```

```
if (open_flag & O_CREAT)
    inode_lock(dir->d_inode);
else
    inode_lock_shared(dir->d_inode);
error = lookup_open(nd, &path, file, op, got_write, opened);
if (open_flag & O_CREAT)
    inode_unlock(dir->d_inode);
else
    inode_unlock_shared(dir->d_inode);
```

lookup_open

```
/*
 * Look up and maybe create and open the last component.
 *
 * Must be called with i_mutex held on parent.
 *
 * Returns 0 if the file was successfully atomically created (if necessary) and
 * opened.  In this case the file will be returned attached to @file.
 *
 * Returns 1 if the file was not completely opened at this time, though lookups
 * and creations will have been performed and the dentry returned in @path will
 * be positive upon return if O_CREAT was specified.  If O_CREAT wasn't
 * specified then a negative dentry may be returned.
 *
 * An error code is returned otherwise.
 *
 * FILE_CREATE will be set in @*opened if the dentry was created and will be
 * cleared otherwise prior to returning.
 */
static int lookup_open(struct nameidata *nd, struct path *path,
            struct file *file,
            const struct open_flags *op,
            bool got_write, int *opened)
```

lookup_open

```
/* Negative dentry, just create the file */
if (!dentry->d_inode && (open_flag & O_CREAT)) {
    *opened |= FILE_CREATED;
    audit_inode_child(dir_inode, dentry, AUDIT_TYPE_CHILD_CREATE);
    if (!dir_inode->i_op->create) {
        error = -EACCES;
        goto out_dput;
    }
    error = dir_inode->i_op->create(dir_inode, dentry, mode,
                    open_flag & O_EXCL);
    if (error)
        goto out_dput;
    fsnotify_create(dir_inode, dentry);
}
if (unlikely(create_error) && !dentry->d_inode) {
    error = create_error;
    goto out_dput;
}
```

inode_operations

- Defined in namei.c

```
const struct inode_operations ext2_dir_inode_operations = {
    .create     = ext2_create,
    .lookup     = ext2_lookup,
    .link       = ext2_link,
    .unlink     = ext2_unlink,
    .symlink    = ext2_symlink,
    .mkdir      = ext2_mkdir,
    .rmdir      = ext2_rmdir,
    .mknod      = ext2_mknod,
    .rename     = ext2_rename,
#ifdef CONFIG_EXT2_FS_XATTR
    .listxattr  = ext2_listxattr,
#endif
    .setattr    = ext2_setattr,
    .get_acl    = ext2_get_acl,
    .set_acl    = ext2_set_acl,
    .tmpfile    = ext2_tmpfile,
};
```

Creating a inode

- **ext2_create()**
  - Defined in namei.c

```
/*
 * By the time this is called, we already have created
 * the directory cache entry for the new file, but it
 * is so far negative - it has no inode.
 *
 * If the create succeeds, we fill in the inode information
 * with d_instantiate().
 */
static int ext2_create (struct inode * dir, struct dentry * dentry, umode_t mode, bool excl)
```

- Parameters
  - dir : inode of directory
  - dentry : corresponding directory entry with this file
  - mode: access mode

dentry

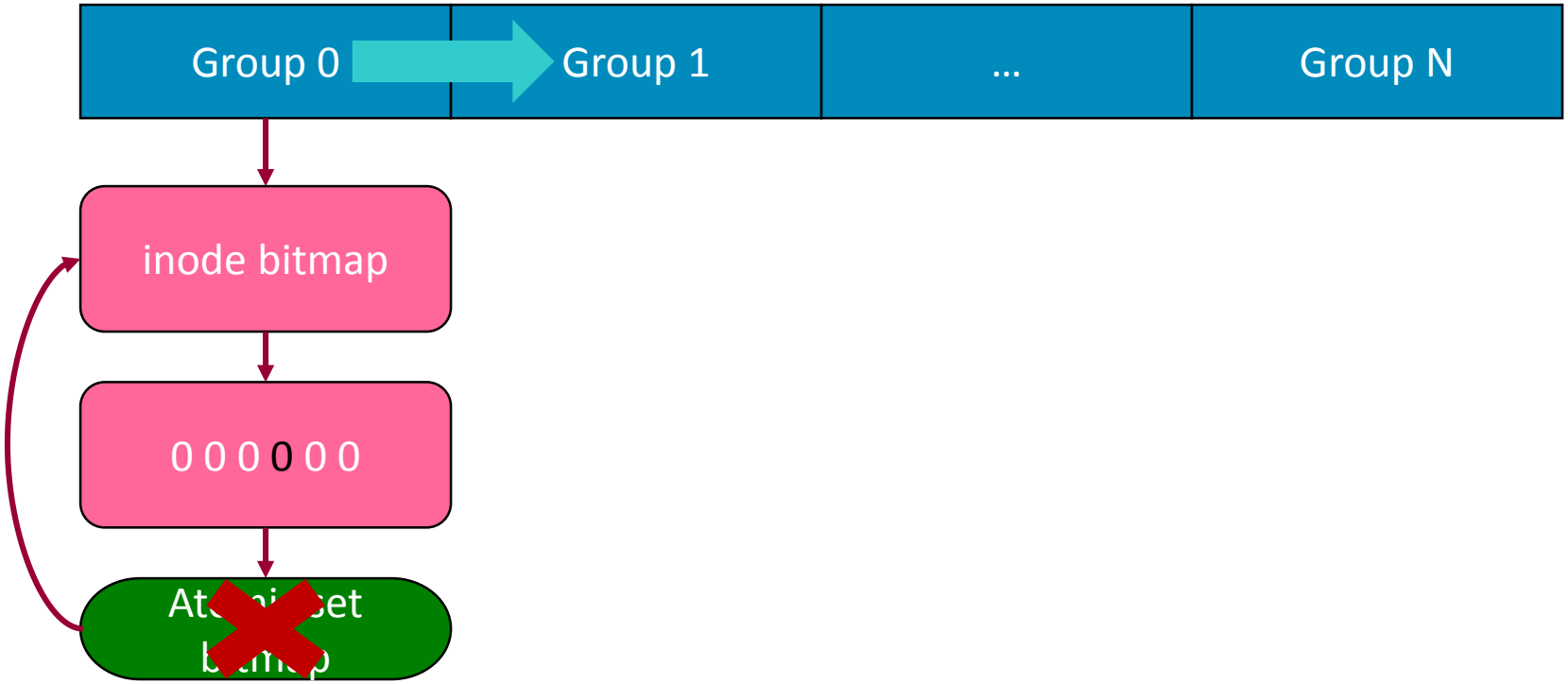
```
struct dentry {
    /* RCU lookup touched fields */
    unsigned int d_flags;       /* protected by d_lock */
    seqcount_t d_seq;       /* per dentry seqlock */
    struct hlist_bl_node d_hash;    /* lookup hash list */
    struct dentry *d_parent;    /* parent directory */
    struct qstr d_name;
    struct inode *d_inode;      /* Where the name belongs to - NULL is
                                 * negative */
    unsigned char d_iname[DNAME_INLINE_LEN];    /* small names */

    /* Ref lookup also touches following */
    struct lockref d_lockref;   /* per-dentry lock and refcount */
    const struct dentry_operations *d_op;
    struct super_block *d_sb;   /* The root of the dentry tree */
    unsigned long d_time;       /* used by d_revalidate */
    void *d_fsdata;         /* fs-specific data */

    union {
        struct list_head d_lru;     /* LRU list */
        wait_queue_head_t *d_wait;  /* in-lookup ones only */
    };
    struct list_head d_child;   /* child of parent list */
    struct list_head d_subdirs; /* our children */
    /*
     * d_alias and d_rcu can share memory
     */
    union {
        struct hlist_node d_alias;  /* inode alias list */
        struct hlist_bl_node d_in_lookup_hash;  /* only for in-lookup ones */
        struct rcu_head d_rcu;
    } d_u;
};
```

ext2_create

```
static int ext2_create (struct inode * dir, struct dentry * dentry, umode_t mode, bool excl)
{
    struct inode *inode;
    int err;

    err = dquot_initialize(dir);
    if (err)
        return err;

    inode = ext2_new_inode(dir, mode, &dentry->d_name);
    if (IS_ERR(inode))
        return PTR_ERR(inode);

    inode->i_op = &ext2_file_inode_operations;
    if (test_opt(inode->i_sb, NOBH)) {
        inode->i_mapping->a_ops = &ext2_nobh_aops;
        inode->i_fop = &ext2_file_operations;
    } else {
        inode->i_mapping->a_ops = &ext2_aops;
        inode->i_fop = &ext2_file_operations;
    }
    mark_inode_dirty(inode);
    return ext2_add_nondir(dentry, inode);
}
```

Allocating a inode

- **`ext2_new_inode()`**
  - ialloc.c
- Work flow



Allocated a inode

- Mark bitmap dirty
- Fill in the inode
  - ino(not accessed)
  - Timestamp
  - Mode
- Mark inode dirty

- Note
  - Directory and file have no difference when allocating a inode

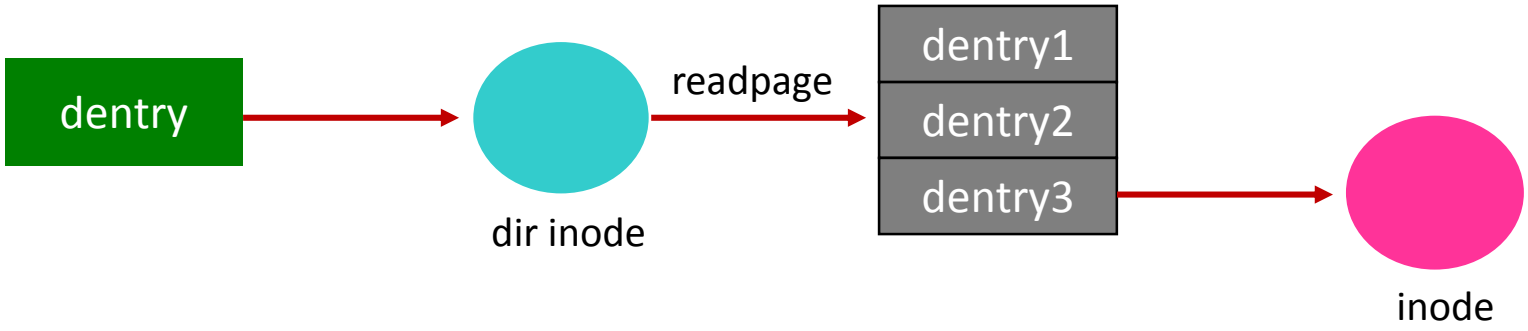
Add link

```
static int ext2_create (struct inode * dir, struct dentry * dentry, umode_t mode, bool excl)
{
    struct inode *inode;
    int err;

    err = dquot_initialize(dir);
    if (err)
        return err;

    inode = ext2_new_inode(dir, mode, &dentry->d_name);
    if (IS_ERR(inode))
        return PTR_ERR(inode);

    inode->i_op = &ext2_file_inode_operations;
    if (test_opt(inode->i_sb, NOBH)) {
        inode->i_mapping->a_ops = &ext2_nobh_aops;
        inode->i_fop = &ext2_file_operations;
    } else {
        inode->i_mapping->a_ops = &ext2_aops;
        inode->i_fop = &ext2_file_operations;
    }
    mark_inode_dirty(inode);
    return ext2_add_nondir(dentry, inode);
}
```

Add link

- Parent – child
- `ext2_add_nondir` → `ext2_add_link`
- **`ext2_add_link()`**
  - Defined in dir.c

```
int ext2_add_link (struct dentry *dentry, struct inode *inode)
```



Open is finished

Write

- write() writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd.
  - `ssize_t write(int fd, const void *buf, size_t count);`
- System call
  - <fs/read_write.c>
  - `Vfs → ext2`
    - `vfs_write → __vfs_write → new_sync_write → filp->f_op->write_iter → ext2_file_write_iter → generic_file_write_iter → __generic_file_write_iter → generic_perform_write`

generic_perform_write

```
status = a_ops->write_begin(file, mapping, pos, bytes, flags,
                &page, &fsdata);//为数据写入分配页缓存并为这个page准备
                                //一组buffer_head结构用于描述组成这个page的数据块
if (unlikely(status < 0))
    break;

if (mapping_writably_mapped(mapping))
    flush_dcache_page(page);

copied = iov_iter_copy_from_user_atomic(page, i, offset, bytes);//将用户空间数据拷入该页缓存
flush_dcache_page(page);

status = a_ops->write_end(file, mapping, pos, bytes, copied,
                page, fsdata);//将page中的每一个buffer_head标记为dirty
```

Address_space_operations

```
const struct address_space_operations ext2_aops = {
    .readpage        = ext2_readpage,
    .readpages       = ext2_readpages,
    .writepage       = ext2_writepage,
    .write_begin         = ext2_write_begin,
    .write_end       = ext2_write_end,
    .bmap            = ext2_bmap,
    .direct_IO       = ext2_direct_IO,
    .writepages      = ext2_writepages,
    .migratepage         = buffer_migrate_page,
    .is_partially_uptodate  = block_is_partially_uptodate,
    .error_remove_page  = generic_error_remove_page,
};
```

ext2_write_begin

```
static int
ext2_write_begin(struct file *file, struct address_space *mapping,
        loff_t pos, unsigned len, unsigned flags,
        struct page **pagep, void **fsdata)
{
    int ret;

    ret = block_write_begin(mapping, pos, len, flags, pagep,
                ext2_get_block);
    if (ret < 0)
        ext2_write_failed(mapping, pos + len);
    return ret;
}
```

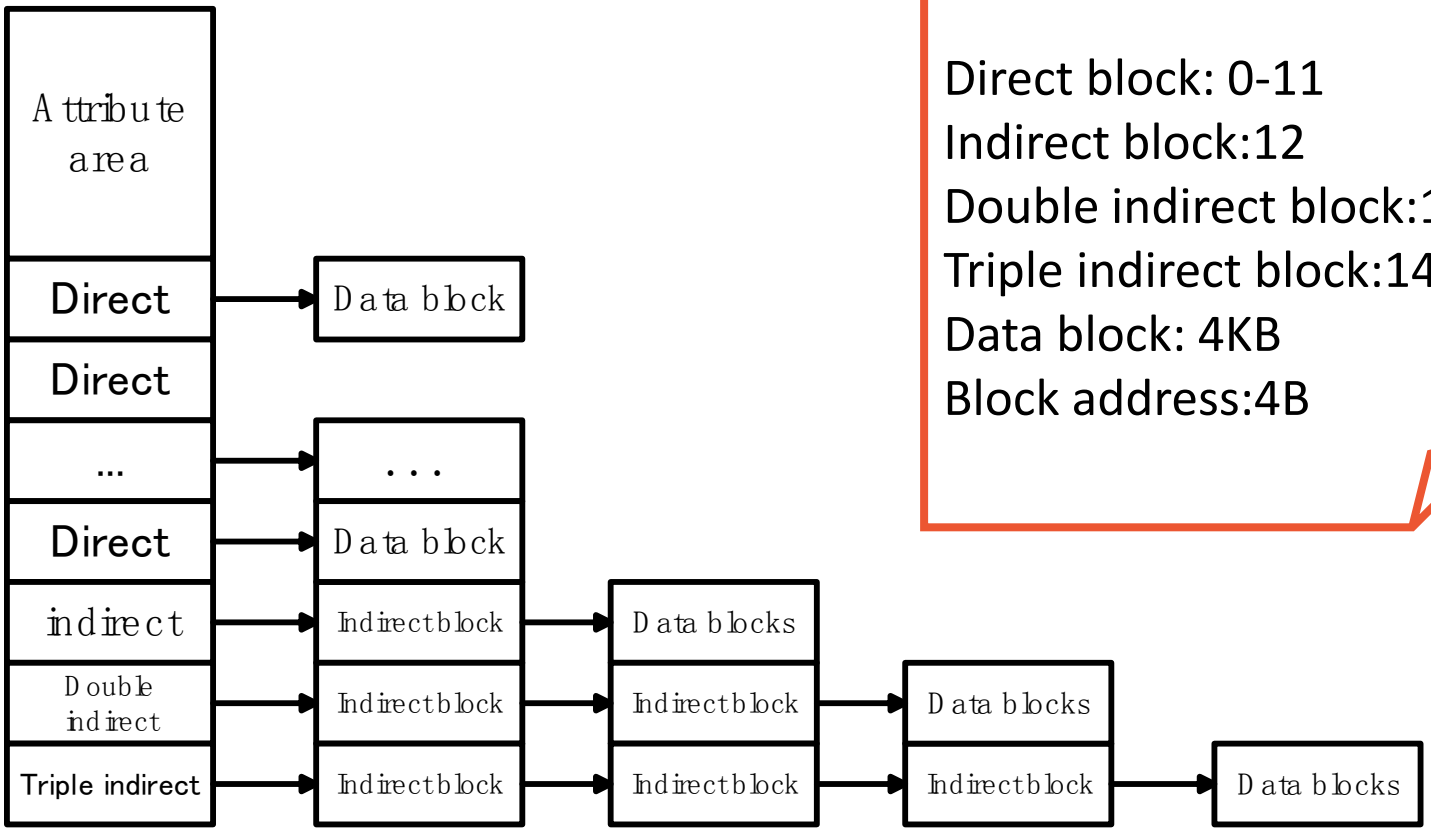
ext2_get_block

- `ext2_get_block → ext2_get_blocks`
  - Defined in inode.c

```
 * return > 0, # of blocks mapped or allocated.
 * return = 0, if plain lookup failed.
 * return < 0, error case.

static int ext2_get_blocks(struct inode *inode,
                sector_t iblock, unsigned long maxblocks,
                u32 *bno, bool *new, bool *boundary,
                int create)
{
    int err;
    int offsets[4];
    Indirect chain[4];
    Indirect *partial;
    ext2_fsblk_t goal;
    int indirect_blks;
    int blocks_to_boundary = 0;
    int depth;
    struct ext2_inode_info *ei = EXT2_I(inode);
    int count = 0;
    ext2_fsblk_t first_block = 0;

    BUG_ON(maxblocks == 0);

    depth = ext2_block_to_path(inode,iblock,offsets,&blocks_to_boundary);
```

ext2_block_to_path

- parse the block number into array of offsets
- Review



Direct block: 0-11
Indirect block:12
Double indirect block:13
Triple indirect block:14
Data block: 4KB
Block address:4B

ext2_block_to_path

- Return the depth
  - 1: direct block
  - 2: indirect block
  - 3: double indirect block
  - 4: triple indirect block

```
depth = ext2_block_to_path(inode,iblock,offsets,&blocks_to_boundary);
```

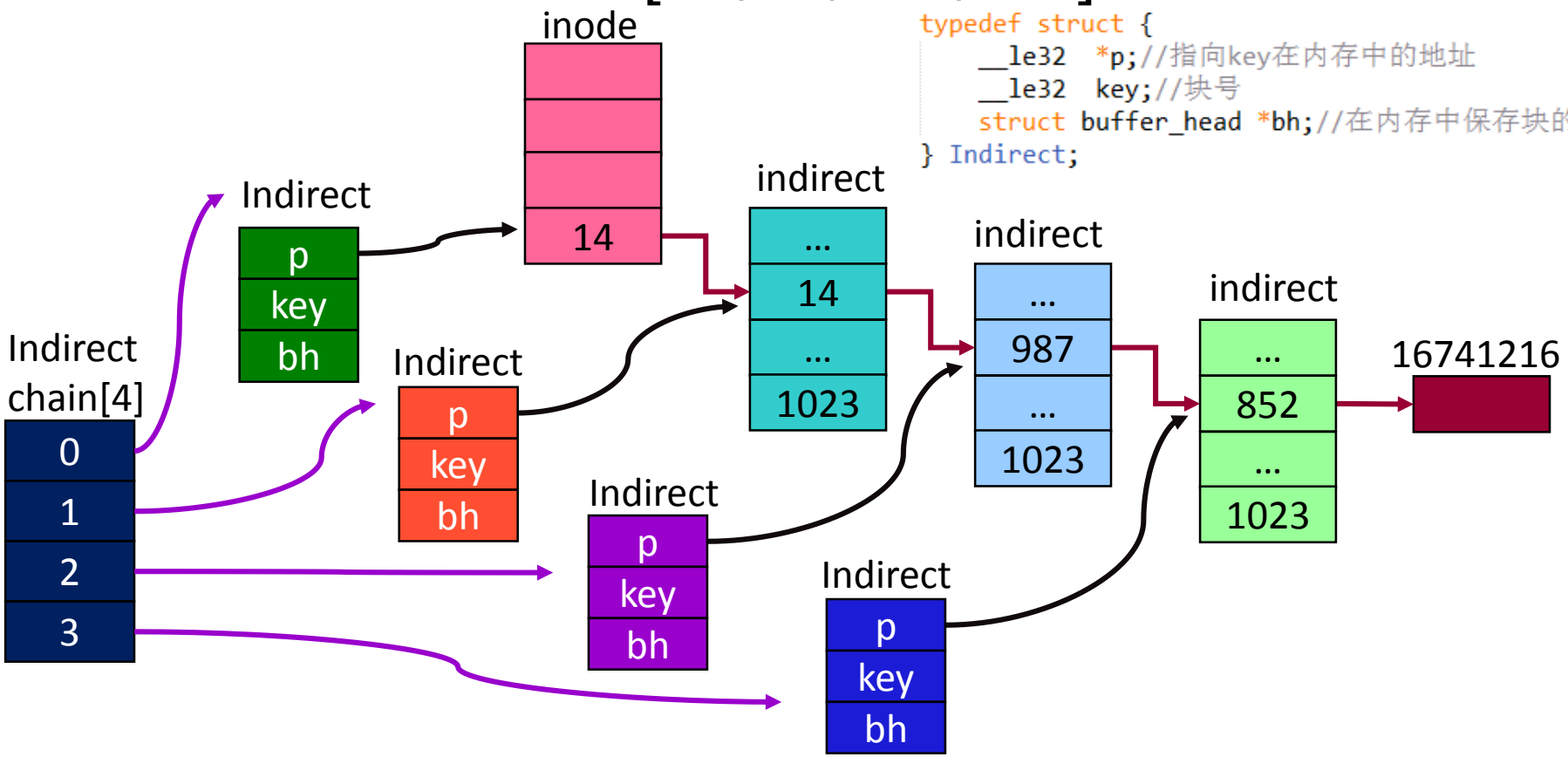
- Fill in the offset array

Block number	Offset[0]	Offset[1]	Offset[2]	Offset[3]	Depth
10	10	0	0	0	1
14	12	2	0	0	2
1049	13	0	13	0	3
16741216	14	14	987	852	4

Offsets – indirect chain

- **`ext2_get_branch()`**
- block 16741216:[14,14,987,852]

```
typedef struct {
    __le32  *p;//指向key在内存中的地址
    __le32  key;//块号
    struct buffer_head *bh;//在内存中保存块的数据
} Indirect;
```



Block allocation

- After location branch, we need to do block allocation

```
if (S_ISREG(inode->i_mode) && (!ei->i_block_alloc_info))
    ext2_init_block_alloc_info(inode);//初始化预分配结构
```

- `ext2_init_block_alloc_info()`
  - Defined in balloc.c
- Allocate and initialize the reservation window structure, and link the window to the ext2 inode structure at last
- The reservation window structure is only dynamically allocated and linked to ext2 inode the first time the open file needs a new block.

Goal block

- `ext2_find_goal()`
  - Find a goal block to alocate

1. Continue to last allocated block

- `ext2_find_near()`
  - Find a near block

2. Near to indirect block
3. Put it into the same cylinder group

Block allocation

- Counting total number of blocks, including the direct and indirect blocks

```
/* the number of blocks need to allocate for [d,t]indirect blocks */
indirect_blks = (chain + depth) - partial - 1;
/*
 * Next look up the indirect map to count the totoal number of
 * direct blocks to allocate for this branch.
 */
count = ext2_blks_to_allocate(partial, indirect_blks,
                    maxblocks, blocks_to_boundary);//统计一共需要分配的block数目
/*
 * XXX ???? Block out ext2_truncate while we alter the tree
 */
err = ext2_alloc_branch(inode, indirect_blks, &count, goal,
                offsets + (partial - chain), partial);
```

Block allocation

- **ext2_alloc_branch()**

```
static int ext2_alloc_branch(struct inode *inode,
                int indirect_blks, int *blks, ext2_fsblk_t goal,
                int *offsets, Indirect *branch)
{
    int blocksize = inode->i_sb->s_blocksize;
    int i, n = 0;
    int err = 0;
    struct buffer_head *bh;
    int num;
    ext2_fsblk_t new_blocks[4];
    ext2_fsblk_t current_block;

    num = ext2_alloc_blocks(inode, goal, indirect_blks,
                *blks, new_blocks, &err);
```

Block allocation

```
static int ext2_alloc_blocks(struct inode *inode,
            ext2_fsblk_t goal, int indirect_blks, int blks,
            ext2_fsblk_t new_blocks[4], int *err)
{
    int target, i;
    unsigned long count = 0;
    int index = 0;
    ext2_fsblk_t current_block = 0;
    int ret = 0;

    /*
     * Here we try to allocate the requested multiple blocks at once,
     * on a best-effort basis.
     * To build a branch, we should allocate blocks for
     * the indirect blocks(if not allocated yet), and at least
     * the first direct block of this branch.  That's the
     * minimum number of blocks need to allocate(required)
     */
    target = blks + indirect_blks;

    while (1) {
        count = target;
        /* allocating blocks for indirect blocks and direct blocks */
        current_block = ext2_new_blocks(inode,goal,&count,err);
```

Core block allocation function

- **`ext2_new_blocks()`** -- block(s) allocation core function
  - @inode: file inode
  - @goal: given target block(filesystem wide)
  - @count: target number of blocks to allocate
  - @errp: error code

```
ext2_fsblk_t ext2_new_blocks(struct inode *inode, ext2_fsblk_t goal,
                 unsigned long *count, int *errp)
```

- Uses a goal block to assist allocation

ext2_new_blocks

1. Reservation window is live?
2. Fs has free blocks?
3. Goal block is free?
   - No: goal block = first block
   - Compute group no
4. Read group descriptor and block bitmap
5. If free blocks < window size?
   - Yes: window = NULL
6. `ext2_try_to_allocate_with_rsv()` is OK?
   - Yes: return
   - No: find other groups and retry

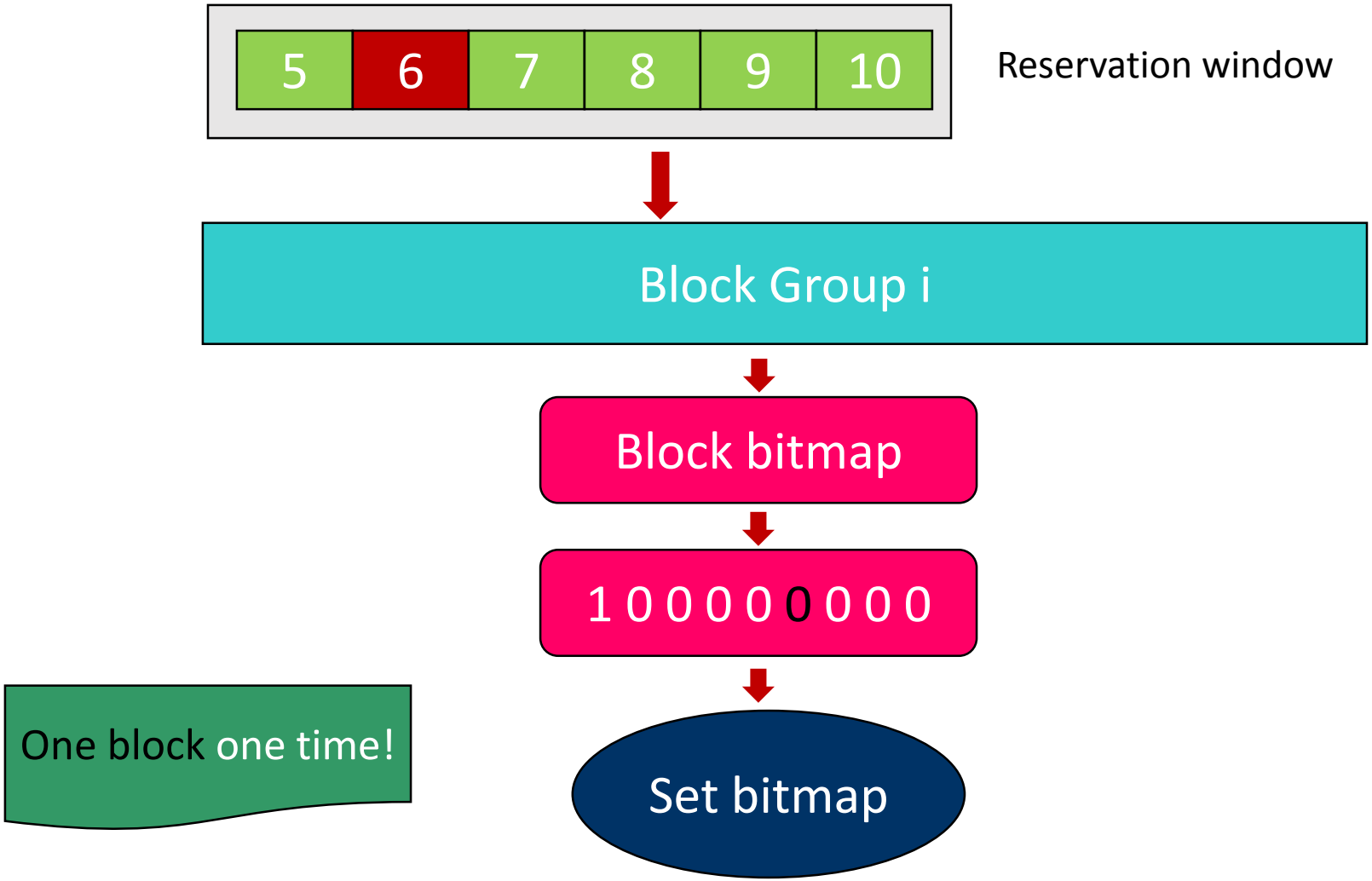
ext2_try_to_allocate_with_rsv

- This is the main function used to allocate a new block and its reservation window

1. If don’t use reservation window
   - Allocate block directly
2. If window is null or goal block is not in window
   - Allocate a new reservation window
3. If window size is small
   - Extend reservation window
4. Allocate block from window

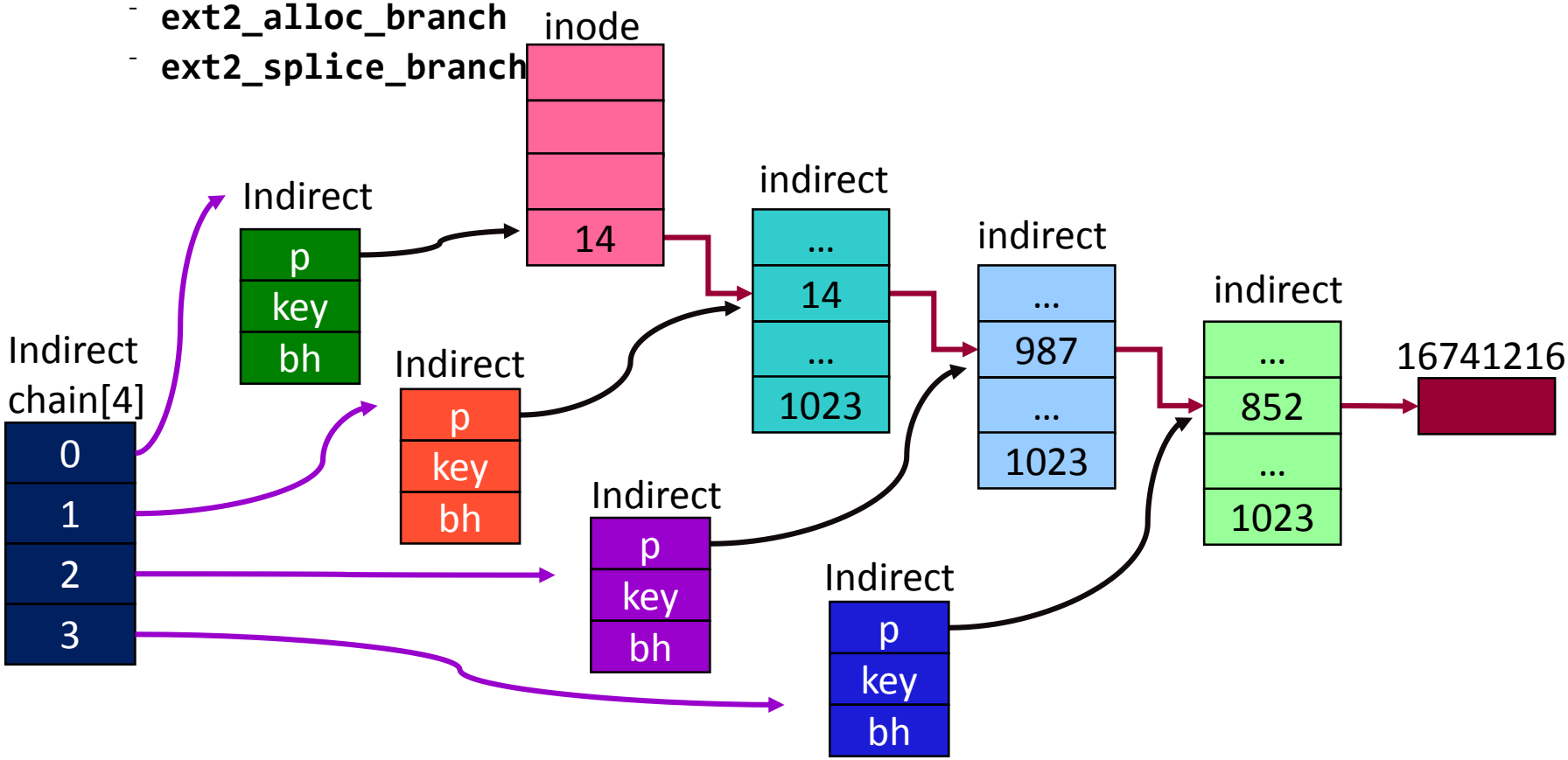
`ext2_try_to_allocate`

ext2_try_to_allocate



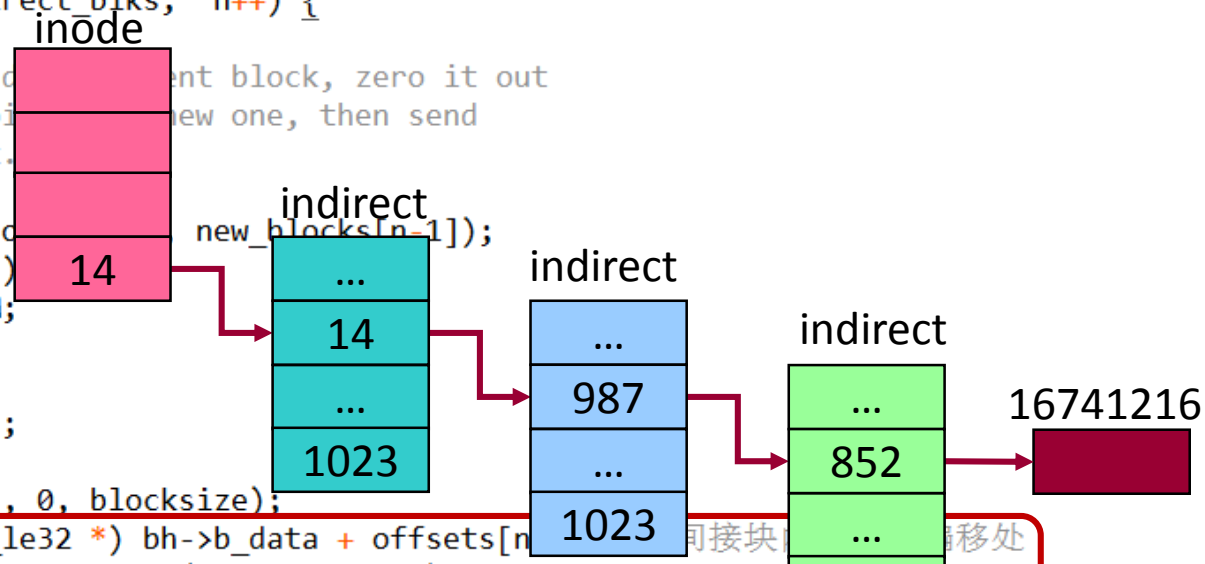
Build links

- After block allocated, build the links between the indirect blocks and data block
  - **`ext2_alloc_branch`**
  - **`ext2_splice_branch`**



ext2_alloc_branch – indirect link

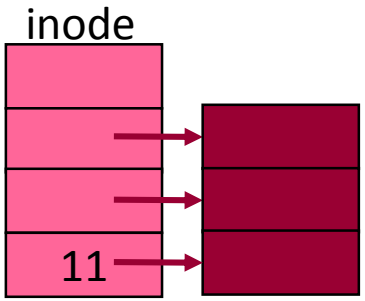
```
for (n = 1; n <= indirect_blks;  n++) {
    /*
     * Get buffer_head                    ent block, zero it out
     * and set the poi                    new one, then send
     * parent to disk.
     */
    bh = sb_getblk(inc                    new_blocks[n-1]);
    if (unlikely(!bh))
        err = -ENOMEM;
        goto failed;
    }
    branch[n].bh = bh;
    lock_buffer(bh);
    memset(bh->b_data, 0, blocksize);
    branch[n].p = (__le32 *) bh->b_data + offsets[n                    间接块                    移处
    branch[n].key = cpu_to_le32(new_blocks[n]);
    *branch[n].p = branch[n].key;//在该偏移处写上下一个间接块的块号
    if ( n == indirect_blks) {//建立一级间接块和数据块的链接
        current_block = new_blocks[n];
        /*
         * End of chain, update the last new metablock of
         * the chain to point to the new allocated
         * data blocks numbers
         */
        for (i=1; i < num; i++)
            *(branch[n].p + i) = cpu_to_le32(++current_block);//因为连续分配
    }
```



ext2_splice_branch – direct link

- No indirect blocks

```
*where->p = where->key;

/*
 * Update the host buffer_head or inode to point to more just allocated
 * direct blocks blocks
 */
if (num == 0 && blks > 1) {
    current_block = le32_to_cpu(where->key) + 1;
    for (i = 1; i < blks; i++)
        *(where->p + i ) = cpu_to_le32(current_block++);
}
```



Write

- write_begin
  - ext2_write_begin
    - ext2_get_block
      - ext2_get_blocks
        - ext2_block_to_path
        - ext2_get_branch
        - ext2_blks_to_allocate
        - ext2_alloc_branch
          - ext2_alloc_blocks
            - ext2_new_blocks
              - ext2_try_to_allocate_with_rsv
                - ext2_try_to_allocate
- write_begin

Read

- read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf
  - `ssize_t read(int fd, void *buf, size_t count);`
- System call
  - `vfs_read → __vfs_read → ext2_file_read_iter →generic_file_read_iter → do_generic_file_read → ext2_readpage →mpage_readpage →do_mpage_readpage`

```
static int ext2_readpage(struct file *file, struct page *page)
{
    return mpage_readpage(page, ext2_get_block);
}

bio = do_mpage_readpage(bio, page, 1, &last_block_in_bio,
        &map_bh, &first_logical_block, get_block, gfp);
```

Read - ext2_get_blocks

```
got_it:
    if (count > blocks_to_boundary)
        *boundary = true;
    err = count;
    /* Clean up and exit */
    partial = chain + depth - 1;    /* the whole chain */
cleanup:
    while (partial > chain) {
        brelse(partial->bh);
        partial--;
    }
    if (err > 0)
        *bno = le32_to_cpu(chain[depth-1].key);
    return err;
```

Sync

- Forced write file from memory into disk

- `do_fsync`
- `vfs_fsync`
- `vfs_fsync_range`
- `ext2_fsync`
- `generic_file_fsync`
- `__generic_file_fsync`

__generic_file_fsync

```
int __generic_file_fsync(struct file *file, loff_t start, loff_t end,
                         int datasync)
{
    struct inode *inode = file->f_mapping->host;
    int err;
    int ret;

    //sync file address_space
    err = filemap_write_and_wait_range(inode->i_mapping, start, end);
    if (err)
        return err;

    inode_lock(inode);
    //sync file associate mapping
    ret = sync_mapping_buffers(inode->i_mapping);
    if (!(inode->i_state & I_DIRTY_ALL))
        goto out;
    if (datasync && !(inode->i_state & I_DIRTY_DATASYNC))
        goto out;

    //sync metadata
    err = sync_inode_metadata(inode, 1);
    if (ret == 0)
        ret = err;

out:
    inode_unlock(inode);
    return ret;
}
```

filemap_write_and_wait_range

- write out & wait on a file range
  - @mapping: the address_space for the pages
  - @lstart:offset in bytes where the range starts
  - @lend:offset in bytes where the range ends (inclusive)

- `__filemap_fdatawrite_range`
- do_writepages

```
if (mapping->a_ops->writepages)
        ret = mapping->a_ops->writepages(mapping, wbc);
```

- ext2_writepages

```
return mpage_writepages(mapping, wbc, ext2_get_block);
```

Other operations

- File operaions
  - Llseek
  - Mmap
- Inode operations
  - Listxattr
  - Setattr
  - Get_acl
  - Set_acl
- Address_space operations
  - Direct_IO
  - Readpage
- Directory operations
  - Read(ls)

Thanks for your attention!