EXT4读写流程

EXT4文件系统格式化命令如下：

mkfs.ext4 -O ^extent,^has\_journal -v -L data 16M.img

EXT4文件feature的说明：

filetype:目录的数据结构中包含了文件类型，因此文件名的长度为:name\_len & 0xFF

flex\_bg :块组描述符中的meta-data（block/inode 位图）可以放在任何位置，某个块组的描述符不必限定在某个块组中

sparse\_super：减少超级块和组描述符的备份，只备份在1,3,5,7,…3^n,5^n,7^n的块组中

huge\_file：

uninit\_b：格式化EXT4时不全部初始化块组描述符，代码中需要就是你bg\_checksum

has\_journal：文件系统的日志,EXT4在格式化时可以去除日志功能

resize\_inode：为GDT留有一定的保留块

**在root目录下创建dob文件夹**

mkdir /dob

**1.读取文件系统的superblock，获得文件系统的基本信息：**

1) Feature: filetype flex\_bg sparse\_super huge\_file uninit\_bg etc.

2)Block size,inode size & Block count, inode count

3)计算块组数gd\_num: (s\_blocks\_count+EXT2\_BLOCK\_SIZE(super)) / s\_blocks\_per\_group

**2.读取gd\_num个快组描述符**

1.获得块位图的位置：bg\_block\_bitmap

2获得inode位图的位置：bg\_inode\_bitmap

3.获得inode\_table的位置：bg\_inode\_table

**3.找到空闲的的block/inode号 &初始化目录块**

1.根据块位图找到空闲的block

2.初始化目录块建立基本的文件夹(“.” &&”..”)

3.根据inode位图找到空闲的inode号

4.根据inode号找到inode table的位置并初始化inode节点

**4.填充root的目录块**

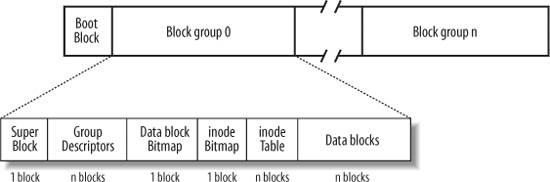
1.新建一个struct ext2\_dir\_entry\_t变量，填充它并更新到父目录中

**5.更新block/inode位图，更新inode table**

1.更新相应的块组描述符

2.更新superblock

EXT文件系统的磁盘结构图：



创建目录的代码如下：

/\* 更新父目录的目录块将dob文件夹加入其中 \*/

void update\_parent\_block(int inode\_table\_loc, struct ext2\_dir\_entry\_2 \*new\_dir)

{

int retval = 0;

struct ext2\_inode inode;

char \*buf = malloc(1024);

retval = io\_channel\_read\_blk64(current\_fs->io, inode\_table\_loc, 1, buf);

memcpy(&inode, buf + current\_fs->super->s\_inode\_size \* (EXT2\_ROOT\_INO - 1), 128);

retval = io\_channel\_read\_blk64(current\_fs->io, inode.i\_block[0], 1, buf);

//list files under root

struct ext2\_dir\_entry\_2 \*dirp;

int offset = 0;

char \*charp = buf;

while(1) {

dirp = (struct ext2\_dir\_entry\_2 \*)charp;

if (dirp->name\_len > 0) {

printf("file name:%15s ino:%5d, rec\_len:%d\n", dirp->name, dirp->inode, dirp->rec\_len);

offset = offset + dirp->rec\_len;

if ((offset == 1024) && ((8+dirp->name\_len) != dirp->rec\_len)) {

offset = offset - dirp->rec\_len;

printf("[struct len: %d, rec\_len:%d]\n", 8 + dirp->name\_len, dirp->rec\_len);

break;

}

}

charp = charp + dirp->rec\_len;

}

if ((8 + dirp->name\_len) <= 12)

dirp->rec\_len = 12;

else

dirp->rec\_len = ( (8 + dirp->name\_len + 3) / 4 ) << 2; **//!!rec\_len %4 must is zero**

offset += dirp->rec\_len;

charp = charp + dirp->rec\_len;

memcpy(charp, new\_dir, new\_dir->rec\_len);

dirp = (struct ext2\_dir\_entry\_2 \*)charp;

dirp->rec\_len = 1024 - offset;

/\* update dir data \*/

io\_channel\_write\_blk64(current\_fs->io, inode.i\_block[0], 1, buf);

}

\_\_u16 ext2fs\_csum(ext2\_filsys fs, struct ext2\_group\_desc \*desc, dgrp\_t group);

/\* mkdir /dob dir \*/

void do\_b(int argc, char \*argv[])

{

struct ext2\_super\_block \*super;

super = current\_fs->super;

struct ext2\_group\_desc \*gd;

char \*buf;

int retval = 0, i = 0, bits = 0;

char \*block\_bitmap = malloc(EXT2\_BLOCK\_SIZE(super));

char \*inode\_bitmap = malloc(EXT2\_BLOCK\_SIZE(super));

current\_fs->io->flags |= CHANNEL\_FLAGS\_WRITETHROUGH;

int gd\_num = (super->s\_blocks\_count + EXT2\_BLOCK\_SIZE(super)) /

super->s\_blocks\_per\_group;

/\* allocate Group desc \*/

gd = (struct ext2\_group\_desc \*)malloc(gd\_num \* EXT2\_DESC\_SIZE(super));

buf = malloc(EXT2\_BLOCK\_SIZE(super));

memset(buf, 0, 1024);

/\* Read Group desc \*/

retval = io\_channel\_read\_blk64(current\_fs->io, 2, 1, buf);

for (i = 0; i < gd\_num; i++) {

memcpy((char \*)&gd[i], buf + EXT2\_DESC\_SIZE(super) \* i, EXT2\_DESC\_SIZE(super));

printf("Group [%d] (%5d ~ %5d) >> ",

i, 1 + EXT2\_BLOCKS\_PER\_GROUP(super) \* i, EXT2\_BLOCKS\_PER\_GROUP(super) \* (i + 1));

printf("Block bitmap:%4d, inode bitmap:%4d, inode table:%4d,"

" free block:%4d, free inode:%4d\n",

gd[i].bg\_block\_bitmap, gd[i].bg\_inode\_bitmap, gd[i].bg\_inode\_table,

gd[i].bg\_free\_blocks\_count, gd[i].bg\_free\_inodes\_count);

}

/\* find & allocate block \*/

retval = io\_channel\_read\_blk64(current\_fs->io, gd[0].bg\_block\_bitmap, 1, buf);

int data\_block = 0;

for (i = 0; i < EXT2\_BLOCK\_SIZE(super); i++) {

for (bits = 0; bits < 8; bits++) {

if ((buf[i] & (1 << bits)) == 0) {

buf[i] = buf[i] | (1 << bits);

data\_block = i \* 8 + bits + 1; **/\* Group 0: start block & ino number is 1 \*/**

goto find\_block;

}

}

}

find\_block:

retval = io\_channel\_write\_blk64(current\_fs->io, gd[0].bg\_block\_bitmap, 1, buf);

memcpy(block\_bitmap, buf, EXT2\_BLOCK\_SIZE(super)); /\* save block bitmap & update it \*/

printf("Find first unused block @%d\n", data\_block);

/\* Get a inode ino & fill it \*/

retval = io\_channel\_read\_blk64(current\_fs->io, gd[0].bg\_inode\_bitmap, 1, buf);

int inode\_ino = 0;

for (i = 0; i < EXT2\_BLOCK\_SIZE(super); i++) {

for (bits = 0; bits < 8; bits++) {

if ((buf[i] & (1 << bits)) == 0) {

buf[i] = buf[i] | (1 << bits);

inode\_ino = i \* 8 + bits + 1;

goto find\_inode;

}

}

}

find\_inode:

retval = io\_channel\_write\_blk64(current\_fs->io, gd[0].bg\_inode\_bitmap, 1, buf);

memcpy(inode\_bitmap, buf, EXT2\_BLOCK\_SIZE(super)); /\* save inode bitmap & update it \*/

printf("Find first unused ino @%d\n", inode\_ino);

struct ext2\_inode new\_inode;

memset(&new\_inode, 0, sizeof(struct ext2\_inode));

new\_inode.i\_mode = LINUX\_S\_IFDIR | (0755);

new\_inode.i\_uid = new\_inode.i\_gid = 0;

ext2fs\_iblk\_set(current\_fs, &new\_inode, 1);

if (current\_fs->super->s\_feature\_incompat & EXT3\_FEATURE\_INCOMPAT\_EXTENTS)

new\_inode.i\_flags |= EXT4\_EXTENTS\_FL;

else

new\_inode.i\_block[0] = data\_block;

new\_inode.i\_links\_count = 2;

new\_inode.i\_size = current\_fs->blocksize; /\* 1024 block size \*/

/\* update father dir \*/

struct ext2\_dir\_entry new\_dir;

int filetype = 0;

**if (current\_fs->super->s\_feature\_incompat &**

**EXT2\_FEATURE\_INCOMPAT\_FILETYPE) //filetype feature**

**filetype = EXT2\_FT\_DIR << 8;**

new\_dir.inode = inode\_ino;

strncpy(new\_dir.name, "dob\0", 4);

new\_dir.name\_len = 3 | filetype;

new\_dir.rec\_len = 12;

\_\_u32 t = current\_fs->now ? current\_fs->now : time(NULL);

if (!new\_inode.i\_ctime)

new\_inode.i\_ctime = t;

if (!new\_inode.i\_mtime)

new\_inode.i\_mtime = t;

if (!new\_inode.i\_atime)

new\_inode.i\_atime = t;

char \*block\_data;

update\_parent\_block(gd[0].bg\_inode\_table, (struct ext2\_dir\_entry\_2 \*)&new\_dir);

/\* initialize new dir 's data (include . & ..) \*/

retval = ext2fs\_new\_dir\_block(current\_fs, inode\_ino, EXT2\_ROOT\_INO, &block\_data);

io\_channel\_write\_blk64(current\_fs->io, data\_block, 1, block\_data);

/\*update inode table \*/

retval = io\_channel\_read\_blk64(current\_fs->io, gd[0].bg\_inode\_table + (inode\_ino - 1) / 8, 1, buf);

memcpy(buf + 128 \* ((inode\_ino - 1) % 8), &new\_inode, 128);

io\_channel\_write\_blk64(current\_fs->io, gd[0].bg\_inode\_table + (inode\_ino - 1) / 8, 1, buf);

//update Group desc

retval = io\_channel\_read\_blk64(current\_fs->io, 2, 1, buf);

gd[0].bg\_free\_blocks\_count -= 1;

gd[0].bg\_free\_inodes\_count -= 1;

gd[0].bg\_itable\_unused -= 1;

printf("gd[0].bg\_free\_inodes\_count:%d\n", gd[0].bg\_free\_inodes\_count);

gd[0].bg\_checksum = ext2fs\_csum(current\_fs, gd, 0);

printf("csum:0x%x\n", gd[0].bg\_checksum);

memcpy(buf, &gd[0], 32);

io\_channel\_write\_blk64(current\_fs->io, 2, 1, buf);

//update super block

retval = io\_channel\_read\_blk64(current\_fs->io, 1, 1, buf);

struct ext2\_super\_block \*sb = (struct ext2\_super\_block \*)malloc(sizeof(struct ext2\_super\_block));

sb = (struct ext2\_super\_block \*)buf;

sb->s\_free\_blocks\_count -= 1;

sb->s\_free\_inodes\_count -= 1;

io\_channel\_write\_blk64(current\_fs->io, 1, 1, buf);

io\_channel\_flush(current\_fs->io);

free(buf);

free(gd);

return 0;

}