yaffs2文件系统坏块产生记

时间 2014-04-29

原文 http://blog.csdn.net/xingtian19880101/article/details/24691129

对于yaffs2文件系统来说，坏块管理无疑是最关键的问题；下面就Yaffs2文件系统读、写操作来分析坏块产生记。

写操作：

写chunk操作；

参数1：yaffs_dev结构（全局）

参数2：要写的2048字节数据

参数3：这个chunk的oob数据

参数4：是否使用保留区

```
static int yaffs_write_new_chunk(struct yaffs_dev *dev,
        const u8 *data,
        struct yaffs_ext_tags *tags, int use_reserver)
{
            int attempts = 0;
            int write_ok = 0;
            int chunk;

            /*删除checkpt的数据，其实也就是无效掉checkpt的数据*/
            yaffs2_checkpt_invalidate(dev);
            do {
               struct yaffs_block_info *bi = 0;
               int erased_ok = 0;
                /*申请一个没有使用的chunk*/
               chunk = yaffs_alloc_chunk(dev, use_reserver, &bi);
               if (chunk < 0) {
                /*进入到这里表示flash里面已经没有使用空间，没有使用空间，
                整个文件系统进入自读状态*/
                  /* no space */
                  break;
               }

          /* First check this chunk is erased, if it needs
                每个异常关机后再开机后，每个申请块的第一个chunk必须进行异常检查
                这里仅仅是第一个申请到的chunk,除非是强制所有的都要检查，不然只
                检查上面的内容。
                * checking. The checking policy (unless forced
                * always on) is as follows:
                *
                /*检查第一个申请的chunk，如果检查通过，其它的chunk都不用再进行检查
                了；如果检查失败，则再次检查；如果块被擦除，则不需要检查。
```

```
 * Check the first page we try to write in a block.
 * If the check passes then we don't need to check any
 * more. If the check fails, we check again...
 * If the block has been erased, we don't need to check.
 *
 * However, if the block has been prioritised for gc,
 * then we think there might be something odd about
 * this block and stop using it.
 *
 * Rationale: We should only ever see chunks that have
 * not been erased if there was a partially written
 * chunk due to power loss. This checking policy should
 * catch that case with very few checks and thus save a
 * lot of checks that are most likely not needed.
 *
 * Mods to the above
 * If an erase check fails or the write fails we skip the
 * rest of the block.
/* let's give it a try */
attempts++;
 /*如果dev->param.always_check_erased置位，表示所有chunk都检查*/
if (dev->param.always_check_erased)
   bi->skip_erased_check = 0;
 /*第一次的要进行检查*/
if (!bi->skip_erased_check) {
 /*对当前申请的chunk进行检查，检查的原则是，先检查tag数据中的ecc是否正确，
 检查无异常则erased为1，否则erased为0*/
   erased_ok = yaffs_check_chunk_erased(dev, chunk);
   if (erased_ok != YAFFS_OK) {
 /*进入到这里表示数据检查不通过，回收这个异常chunk*/
      yaffs_trace(YAFFS_TRACE_ERROR,
        "**>> yaffs chunk %d was not erased",
        chunk);
      /* If not erased, delete this one,
       * skip rest of block and
       * try another chunk */
 /*下面两个函数目前不想太过考虑*/
      yaffs_chunk_del(dev, chunk, 1, __LINE__);
      yaffs_skip_rest_of_block(dev);
      continue;
   }
}
 /*写数据到chunk，同时写oob区*/
write_ok = yaffs_wr_chunk_tags_nand(dev, chunk, data, tags);
 /*表示不跳过检查*/
if (!bi->skip_erased_check)
   write_ok =
```

```
                        /*这里对写入的数据进行检查，原则是先把main区的数据全部读出来，
                        然后再进行memcmp比较，比较完后再对oob的16个字节yaffs2数据
                        也进行对比，全部正常则表示写正常*/
                       yaffs_verify_chunk_written(dev, chunk, data, tags);
                    if (write_ok != YAFFS_OK) {
                     /*如果写不正常，则进行下面的回收操作,下面对这个问题再另行分析*/
                     yaffs_handle_chunk_wr_error(dev, chunk, erased_ok);
                       continue;
                    }
                     /*置位，这个块只检查一次*/
                    bi->skip_erased_check = 1;
                    /* Copy the data into the robustification buffer */
                    yaffs_handle_chunk_wr_ok(dev, chunk, data, tags);
                } while (write_ok != YAFFS_OK &&
                     (yaffs_wr_attempts <= 0 || attempts <= yaffs_wr_attempts));
                if (!write_ok)
                    chunk = -1;
                if (attempts > 1) {
                    yaffs_trace(YAFFS_TRACE_ERROR,
                        "**>> yaffs write required %d attempts",
                        attempts);
                     /*计算写异常次数*/
                    dev->n_retired_writes += (attempts - 1);
                }
                return chunk;
}

下面接着分析yaffs_handle_chunk_wr_error()函数：
static void yaffs_handle_chunk_wr_error(struct yaffs_dev *dev, int nand_chunk,
                int erased_ok)
{
        int flash_block = nand_chunk / dev->param.chunks_per_block;
        struct yaffs_block_info *bi = yaffs_get_block_info(dev, flash_block);
      /*操作这个块优先进行垃圾回收,下面紧接着分析这个函数*/
        yaffs_handle_chunk_error(dev, bi);
        if (erased_ok) {
            /* Was an actual write failure,
             * so mark the block for retirement.*/
             /*这里置位，直接表示这个块会被标记为坏块*/
            bi->needs_retiring = 1;
            yaffs_trace(YAFFS_TRACE_ERROR | YAFFS_TRACE_BAD_BLOCKS,
              "**>> Block %d needs retiring", flash_block);
        }
        /* Delete the chunk */
        yaffs_chunk_del(dev, nand_chunk, 1, __LINE__);
        yaffs_skip_rest_of_block(dev);
}
```

下面分析yaffs_handle_chunk_err()函数：

```
void yaffs_handle_chunk_error(struct yaffs_dev *dev,
                struct yaffs_block_info *bi)
{
        if (!bi->gc_prioritise) {
            /*表示这个块上有出现过ecc校验出错，
             * 回收时优先回收这一块*/
            bi->gc_prioritise = 1;
            /*表示这个设备上有优先可以回收的块*/
            dev->has_pending_prioritised_gc = 1;
            /*记录ecc错误的次数*/
            bi->chunk_error_strikes++;
             /*如果ecc异常超过3次以上，那么表示这个块也就是坏块了*/
            if (bi->chunk_error_strikes > 3) {
                bi->needs_retiring = 1; /* Too many stikes, so retire */
                /*这里是我要检验的一个点，就是有ecc出错的情况下
                 * 这里会有打印信息*/
                yaffs_trace(YAFFS_TRACE_ALWAYS,
                    "yaffs: Block struck out");
            }
        }
}
```

至上，写操作中已经出现了坏块，打上bi->needs_retiring=1的块将会直接标记为坏块

下面进行具体标记坏块分析：

```
void yaffs_block_became_dirty(struct yaffs_dev *dev, int block_no)
{
```

```
struct yaffs_block_info *bi = yaffs_get_block_info(dev, block_no);
int erased_ok = 0;
int i;
/* If the block is still healthy erase it and mark as clean.
 * If the block has had a data failure, then retire it.
 */
yaffs_trace(YAFFS_TRACE_GC | YAFFS_TRACE_ERASE,
    "yaffs_block_became_dirty block %d state %d %s",
    block_no, bi->block_state,
    (bi->needs_retiring) ? "needs retiring" : "");
/*清除当前最旧的块序列号*/
yaffs2_clear_oldest_dirty_seq(dev, bi);
/*当前块可回收*/
bi->block_state = YAFFS_BLOCK_STATE_DIRTY;
/* If this is the block being garbage collected then stop gc'ing */
/*是不是表示当前的这个块正在进行回收,个人认为这里就是表示当前块正在被回收*/
if (block_no == dev->gc_block)
    dev->gc_block = 0;
/* If this block is currently the best candidate for gc
 * then drop as a candidate */

/*如果这是存储的最脏的擦除块，那么直接丢弃它*/
if (block_no == dev->gc_dirtiest) {
    dev->gc_dirtiest = 0;
    dev->gc_pages_in_use = 0;
}
/*数据有错误在这个块上???*/
/*这个块上出现过三次以上的ecc错误*/
if (!bi->needs_retiring) {
    /*使checkpt无效？*/
    yaffs2_checkpt_invalidate(dev);
    /*调用mtd的nand flash接口擦除*/
    erased_ok = yaffs_erase_block(dev, block_no);
    if (!erased_ok) {
        dev->n_erase_failures++;
        yaffs_trace(YAFFS_TRACE_ERROR | YAFFS_TRACE_BAD_BLOCKS,
          "**>> Erasure failed %d", block_no);
    }
}
     /*上面如果bi->needs_retiring置位了，表示这个块直接被标记为坏块*/
```

```
        /* Verify erasure if needed */
        /*检查是否擦除成功*/
        if (erased_ok &&
            ((yaffs_trace_mask & YAFFS_TRACE_ERASE) ||
             !yaffs_skip_verification(dev))) {
            for (i = 0; i < dev->param.chunks_per_block; i++) {
                if (!yaffs_check_chunk_erased(dev,
                    block_no * dev->param.chunks_per_block + i)) {
                    yaffs_trace(YAFFS_TRACE_ERROR,
                        ">>Block %d erasure supposedly OK, but chunk %d not erased",
                        block_no, i);
                }
            }
        }
        /*如果是这样，表示整个块均不能用了，*/
        if (!erased_ok) {
            /* We lost a block of free space */
            /*能用空间减去一个单位的block*/
            dev->n_free_chunks -= dev->param.chunks_per_block;
             /*下面函数将会直接调用标记坏块接口,下面具体分析这个函数*/
            yaffs_retire_block(dev, block_no);
            yaffs_trace(YAFFS_TRACE_ERROR | YAFFS_TRACE_BAD_BLOCKS,
                "**>> Block %d retired", block_no);
            return;
        }
        /* Clean it up... */
        /*块的状态为空*/
        bi->block_state = YAFFS_BLOCK_STATE_EMPTY;
        /*序列号为空*/
        bi->seq_number = 0;
        /*可用块++*/
        dev->n_erased_blocks++;
        /*这个块中被使用的chunk为 0 */
        bi->pages_in_use = 0;
        bi->soft_del_pages = 0;
        bi->has_shrink_hdr = 0;
        bi->skip_erased_check = 1; /* Clean, so no need to check */
        bi->gc_prioritise = 0;
        bi->has_summary=0;
        /*整个块均处理完成了，把这个块的所有的chunk位清除*/
        yaffs_clear_chunk_bits(dev, block_no);
        yaffs_trace(YAFFS_TRACE_ERASE, "Erased block %d", block_no);
}
```

分析 yaffs_retire_block(dev, block_no)函数，下面内容没有什么好分析的了，直接看代码就能知道什么情况：

```
static void yaffs_retire_block(struct yaffs_dev *dev, int flash_block)
{
```

```
        struct yaffs_block_info *bi = yaffs_get_block_info(dev, flash_block);
        yaffs2_checkpt_invalidate(dev);
        yaffs2_clear_oldest_dirty_seq(dev, bi);
        /*标志为坏块*/
        if (yaffs_mark_bad(dev, flash_block) != YAFFS_OK) {
           if (yaffs_erase_block(dev, flash_block) != YAFFS_OK) {
              yaffs_trace(YAFFS_TRACE_ALWAYS,
                 "yaffs: Failed to mark bad and erase block %d",
                 flash_block);
           } else {
              struct yaffs_ext_tags tags;
              int chunk_id =
                 flash_block * dev->param.chunks_per_block;
              u8 *buffer = yaffs_get_temp_buffer(dev);
              memset(buffer, 0xff, dev->data_bytes_per_chunk);
              memset(&tags, 0, sizeof(tags));
              tags.seq_number = YAFFS_SEQUENCE_BAD_BLOCK;
        if (dev->param.write_chunk_tags_fn(dev, chunk_id -
                           dev->chunk_offset,
                           buffer,
                           &tags) != YAFFS_OK)
                 yaffs_trace(YAFFS_TRACE_ALWAYS,
                    "yaffs: Failed to write bad block marker to block %d",
                    flash_block);
              yaffs_release_temp_buffer(dev, buffer);
           }
        }
        bi->block_state = YAFFS_BLOCK_STATE_DEAD;
        bi->gc_prioritise = 0;
        bi->needs_retiring = 0;
        dev->n_retired_blocks++;
}
```

相关文章

- 1. 文件系统笔记ext4 yaffs2 fat ubi (http://www.voidcn.com/article/p-ecxwsqgm-um.html)
- 2. Yaffs2 文件系统移植 (http://www.voidcn.com/article/p-mvjetjke-gr.html)
- 3. Busybox - Yaffs2文件系统 (http://www.voidcn.com/article/p-akiyyftk-pn.html)
- 4. yaffs2文件系统原理 (http://www.voidcn.com/article/p-zalytfof-ds.html)
- 5. yaffs2文件系统移植 (http://www.voidcn.com/article/p-knlmhmap-et.html)
- 6. 制作yaffs2文件系统 (http://www.voidcn.com/article/p-diteigms-gv.html)
- 7. yaffs2文件系统制作 (http://www.voidcn.com/article/p-xwnznwxm-yh.html)
- 8. yaffs2文件系统制作 . . (http://www.voidcn.com/article/p-ogkwwkzs-ns.html)
- 9. yaffs2文件系统介绍 (http://www.voidcn.com/article/p-kwyalbnt-hm.html)
- 10. cramfs文件系统识别nand坏块 (http://www.voidcn.com/article/p-dziqhiol-bdb.html)
- 更多相关文章... (http://www.voidcn.com/relative/p-vlexehib-bks.html)

相关标签/搜索

yaffs2文件系统制作 (http://www.voidcn.com/tag/yaffs2%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F%E5%88%B6%E4%BD%9C) 系统产生 (http://www.voidcn.com/tag/%E7%B3%BB%E7%BB%9F%E4%BA%A7%E7%94%9F) 文件系统损坏

(http://www.voidcn.com/tag/%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F%E6%8D%9F%E5%9D%8F) yaffs2 (http://www.voidcn.com/tag/yaffs2) 系统损坏 (http://www.voidcn.com/tag/%E7%B3%BB%E7%BB%9F%E6%8D%9F%E5%9D%8F) 坏块 (http://www.voidcn.com/tag/%E5%9D%8F%E5%9D%97) 文件系统 (http://www.voidcn.com/tag/%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F) 文件 系统 (http://www.voidcn.com/tag/%E6%96%87%E4%BB%B6+%E7%B3%BB%E7%BB%9F) 系统文件 (http://www.voidcn.com/tag/%E7%B3%BB%E7%BB%9F%E6%96%87%E4%BB%B6) 产生统计 (http://www.voidcn.com/tag/%E4%BA%A7%E7%94%9F%E7%BB%9F%E8%AE%A1) 坏块 (http://www.voidcn.com/cata/6112011) 文件系统-yaffs2文件系统 (http://www.voidcn.com/cata/2106685) yaffs2 (http://www.voidcn.com/cata/1819605) 文件系统 (http://www.voidcn.com/cata/5759765) 文件系统 (http://www.voidcn.com/cata/2312175) 文件系统 (http://www.voidcn.com/cata/1330396) 文件系统 (http://www.voidcn.com/cata/869389) 文件系统 (http://www.voidcn.com/cata/344760) 文件系统 (http://www.voidcn.com/cata/426939) 文件系统 (http://www.voidcn.com/cata/1075636) COSS文件系统 (http://www.voidcn.com/search/ztnxrz) coss 文件系统 (http://www.voidcn.com/search/zxdejz) ubifs文件系统 (http://www.voidcn.com/search/hbmmph) coss文件系统 (http://www.voidcn.com/search/anhruy) cos 文件系统 (http://www.voidcn.com/search/wyagml) ubuntu文件系统 (http://www.voidcn.com/search/swqxnk) gpfdist文件系统 (http://www.voidcn.com/search/reygvr) mt7688 文件系统 (http://www.voidcn.com/search/ecozmv) 3516a系统文件 (http://www.voidcn.com/search/bavyvz) nanopi2文件系统 (http://www.voidcn.com/search/dbpsmi)

每日一句

每一个你不满意的现在，都有一个你没有努力的曾经。

最新文章

相关文章

ds.html)

- 5. yaffs2文件系统移植 (http://www.voidcn.com/article/p-knlmhmap-et.html)
- 6. 制作yaffs2文件系统 (http://www.voidcn.com/article/p-diteigms-gv.html)
- 7. yaffs2文件系统制作 (http://www.voidcn.com/article/p-xwnznwxm-yh.html)
- 8. yaffs2文件系统制作 . . (http://www.voidcn.com/article/p-ogkwwkzs-ns.html)
- 9. yaffs2文件系统介绍 (http://www.voidcn.com/article/p-kwyalbnt-hm.html)
- 10. cramfs文件系统识别nand坏块 (http://www.voidcn.com/article/p-dziqhiol-bdb.html)

>>更多相关文章<< (http://www.voidcn.com/relative/p-vlexehib-bks.html)

本站公众号 (/contact)

欢迎关注本站公众号,获取更多程序园信息