C语言读取ext2文件系统

李世旺

2016年6月20日

摘要

ext2作为简单的文件系统，我们应该学习它的存储格式。本文就ext2文件系统进行分析并用C语言读取文件。

目录

1 实验目的

用C语言读取ext2文件系统

2 实验原理与方案

ext2文件系统的基本单元是一个个的block，每个block常常是 1KB、2KB、4KB等。文件系统的最前 1KB 留做启动用途，1KB 到 2KB 的地方是 superblock 的所在。下一个 block 开始就是块群描述表。我们把一定量的 block 分割开来，叫做块群。每个块群都放了索引节点表(inode table)，用来寻找文件。块群描述表放了所有 inode table 的地址。每一个目录或文件都有一个 inode 指定。目录看作是一个文件，这个文件包含目录下的所有文件或目录的名字和权限等属性，这些属性所占用的地方大小叫做目录的大小。文件或目录本身不包含自己的名字。 inode table 是按照 inode 的值升序排列的一组 inode。根目录的 inode 一般为 2, 也就是第二项 inode. inode 告诉我们它所指向的文件总共多少字节、占用了多少个 512B 的“小块”以及 15 个直接或间接指向文件的 block. 前 12 个表示直接指向文件数据，第 13 个指向一个 block ,这个 block 里面的号码所指向的 block 直接就是数据。后面依次类推。目录文件里面的项是不定长的，但每项都告诉了自己的长度，这样总是能够找到下一项的所在，相当于链表。

2.1 inode

每一个目录或文件都有一个 inode 指定。根目录的 inode 一般为 2, 也就是第二项 inode. inode 告诉我们它所指向的文件总共多少字节、占用了多少个 512B 的“小块”以及 15 个直接或间接指向文件的 block. 前 12 个表示直接指向文件数据，第 13 个指向一个 block ,这个 block 里面的号码所指向的 block 直接就是数据。后面依次类推。

2.2 目录文件

每一个目录或文件都有一个 inode 指定。目录看作是一个文件，这个文件包含目录下的所有文件或目录的名字和权限等属性，这些属性所占用的地方大小叫做目录的大小。目录文件里面的项是不定长的，但每项都告诉了自己的长度，这样总是能够找到下一项的所在，相当于链表。

3 执行结果与分析

$$R^e = R \otimes R$$

```
gcc -Wall -o ext2.exe ext2.c
./ext2.exe
      s_inodes_count  :  10240
      s_blocks_count  :  40960
    s_r_blocks_count  :  2048
 s_free_blocks_count  :  39156
 s_free_inodes_count  :  10213
  s_first_data_block  :  1
    s_log_block_size  :  0
  s_blocks_per_group  :  8192
             s_mtime  :  0
             s_wtime  :  1369782880
         s_mnt_count   :  3
             s_magic  :  ef53
         block group  :  5
         s_first_ino  :  11
        s_inode_size  :  128
      g_block_bitmap  :  162
      g_inode_bitmap  :  163
       g_inode_table  :  164
```

```
              i_mode  :  41ed
              i_size  :  1024
            i_blocks  :  2
             i_block  :  420
        ***  root dir file details ***
-------------------------------------------------
     inode  :  2           rec_len  :  12
  name_len  :  1         file_type  :  2
 file_name  :  .
-------------------------------------------------
     inode  :  2           rec_len  :  12
  name_len  :  2         file_type  :  2
 file_name  :  ..
-------------------------------------------------
     inode  :  11          rec_len  :  20
  name_len  :  10        file_type  :  2
 file_name  :  lost+found
-------------------------------------------------
     inode  :  4097        rec_len  :  16
  name_len  :  8         file_type  :  2
 file_name  :  Baer_sum
-------------------------------------------------
     inode  :  8193        rec_len  :  20
  name_len  :  10        file_type  :  2
 file_name  :  cohomology
-------------------------------------------------
     inode  :  4099        rec_len  :  20
  name_len  :  9         file_type  :  2
 file_name  :  dimension
-------------------------------------------------
     inode  :  12          rec_len  :  16
```

```
  name_len  :  7         file_type  :  1
 file_name  :  eft.txt
----------------------------------------------
     inode  :  4100        rec_len  :  908
  name_len  :  6         file_type  :  2
 file_name  :  repeat
----------------------------------------------
     inode  :  11          rec_len  :  12
  name_len  :  1         file_type  :  2
 file_name  :  .
----------------------------------------------
     inode  :  2           rec_len  :  1012
  name_len  :  2         file_type  :  2
 file_name  :  ..
----------------------------------------------
        *** regular file at root dir ***
              i_mode  :  81a4
              i_size  :  217
            i_blocks  :  2
             i_block  :  2049
We say that a commutative k-algebra R is essentially of finite type if
it is a localization of a finitely generated k-algebra. If k is noetherian,
this implies that R and $R^e = R \otimes R$ are both noethrian rings.
----------------------------------------------
```

4 详细代码

```
#include<stdio.h>
#define __u32  unsigned long
#define __u16  unsigned short
#define __u8   unsigned  char
```

```
int i;
__u32 regularnode = 0;
__u32 g_block_bitmap;
__u32 g_inode_bitmap;
__u32 g_inode_table;
__u16 i_mode;
__u32 i_size;
__u32 i_blocks;
__u32 i_block;
__u32 inode;
__u16 rec_len;
__u8 name_len;
__u8 file_type;
char name;
struct ext2_super_block
{
    /*00*/__u32 s_inodes_count;

    /* inodes 计 数 */
    __u32 s_blocks_count;

    /* blocks 计 数 */
    __u32 s_r_blocks_count;

    /* 保留的 blocks 计 数 */
    __u32 s_free_blocks_count;

    /* 空 闲 的 blocks 计 数 */
    /*10*/__u32 s_free_inodes_count;

    /* 空 闲 的 inodes 计 数 */
```

```
__u32 s_first_data_block;

/* 第一个数据 block */
__u32 s_log_block_size;

/*20*/__u32 s_blocks_per_group;

/* 每 block group 的 block 数量 */
__u32 s_frags_per_group;

/* 可以忽略 */
__u32 s_inodes_per_group;

/* 每 block group 的 inode 数量 */
__u32 s_mtime;

/* Mount time */
/*30*/__u32 s_wtime;

/* Write time */
__u16 s_mnt_count;

/* Maximal mount count */
__u16 s_magic;

/* Magic 签 名 */
__u16 s_state;

/* File system state */
__u16 s_errors;
```

```
/* Behaviour when detecting errors */
__u16 s_minor_rev_level;

/* minor revision level */
/*40*/__u32 s_lastcheck;

/* time of last check */
__u32 s_checkinterval;

/* max. time between checks */
__u32 s_creator_os;

/* 可以忽略 */
__u32 s_rev_level;

/* Revision level */
/*50*/__u16 s_def_resuid;

/* Default uid for reserved blocks */
__u16 s_def_resgid;

/* Default gid for reserved blocks */
__u32 s_first_ino;

/* First non-reserved inode */
__u16 s_inode_size;

/* size of inode structure */
__u16 s_block_group_nr;

/* block group # of this superblock */
```

```
__u32 s_feature_compat;

/* compatible feature set */
/*60*/__u32 s_feature_incompat;

/* incompatible feature set */
__u32 s_feature_ro_compat;

/* readonly-compatible feature set */
/*68*/__u8 s_uuid[16];

/* 128-bit uuid for volume */
/*78*/
char s_volume_name[16];

/* volume name */
/*88*/
char s_last_mounted[64];

/* directory where last mounted */;

/*C8*/__u32 s_algorithm_usage_bitmap;

/* 可以忽略 */
__u8 s_prealloc_blocks;

/* 可以忽略 */
__u8 s_prealloc_dir_blocks;

/* 可以忽略 */
__u16 s_padding1;
```

```
    /* 可以忽略 */
    /*D0*/__u8 s_journal_uuid[16];

    /* uuid of journal superblock */
    /*E0*/__u32 s_journal_inum;

    /* 日志文件的 inode 号数 */
    __u32 s_journal_dev;

    /* 日志文件的 设备 号 */
    __u32 s_last_orphan;

    /* start of list of inodes to delete */
    /*EC*/ // __u32 s_reserved[197]
    /* 可以忽略 */
} ext2_block;

void Initial(int argc, char** argv)
{
    FILE *fp;
    if (2 != argc)
    {
        printf("\nTips: a.exe data.img");
        return;
    }
    fp = fopen(argv[1], "rb");
    // 定位
    fseek(fp, 1024, SEEK_SET);

    /*00*/
```

```
fread(&(ext2_block.s_inodes_count), 4, 1, fp);

/* inodes 计 数 */
fread(&(ext2_block.s_blocks_count), 4, 1, fp);

/* blocks 计 数 */
fread(&(ext2_block.s_r_blocks_count), 4, 1, fp);

/* 保留的 blocks 计 数 */
fread(&(ext2_block.s_free_blocks_count), 4, 1, fp);

/* 空 闲 的 blocks 计 数 */
/*10*/
fread(&(ext2_block.s_free_inodes_count), 4, 1, fp);

/* 空 闲 的 inodes 计 数 */
fread(&(ext2_block.s_first_data_block), 4, 1, fp);

/* 第一个数据 block */
fread(&(ext2_block.s_log_block_size), 4, 1, fp);

/* block 的大小 */;

fseek(fp, 4, SEEK_CUR);

/* 忽略 s_log_frag_size */;

/*20*/
fread(&(ext2_block.s_blocks_per_group), 4, 1, fp);
/** 每 block group 的 block 数量 */
fread(&(ext2_block.s_frags_per_group), 4, 1, fp);
```

```
/* 可以忽略 */
fread(&(ext2_block.s_inodes_per_group), 4, 1, fp);
/* 每 block group 的 inode 数量 */
fread(&(ext2_block.s_mtime), 4, 1, fp);
/* Mount time */
/*30*/
fread(&(ext2_block.s_wtime), 4, 1, fp);
/* Write time */
fread(&(ext2_block.s_mnt_count), 2, 1, fp);
/* Mount count */;

fseek(fp, 2, SEEK_CUR);

/* 忽略 s_max_mnt_count */;

fread(&(ext2_block.s_magic), 2, 1, fp);

/* Magic 签 名 */
fread(&(ext2_block.s_state), 2, 1, fp);

/* File system state */
fread(&(ext2_block.s_errors), 2, 1, fp);

/* Behaviour when detecting errors */
fread(&(ext2_block.s_minor_rev_level), 2, 1, fp);

/* minor revision level */
/*40*/fread(&(ext2_block.s_lastcheck), 4, 1, fp);

/* time of last check */
fread(&(ext2_block.s_checkinterval), 4, 1, fp);
```

```
/* max. time between checks */
fread(&(ext2_block.s_creator_os), 4, 1, fp);

/* 可以忽略 */
fread(&(ext2_block.s_rev_level), 4, 1, fp);

/* Revision level */
/*50*/fread(&(ext2_block.s_def_resuid), 2, 1, fp);

/* Default uid for reserved blocks */
fread(&(ext2_block.s_def_resgid), 2, 1, fp);

/* Default gid for reserved blocks */
fread(&(ext2_block.s_first_ino), 4, 1, fp);

/* First non-reserved inode */
fread(&(ext2_block.s_inode_size), 2, 1, fp);

/* size of inode structure */
fread(&(ext2_block.s_block_group_nr), 2, 1, fp);

/* block group # of this superblock */
fread(&(ext2_block.s_feature_compat), 4, 1, fp);

/* compatible feature set */
/*60*/fread(&(ext2_block.s_feature_incompat), 4, 1, fp);

/* incompatible feature set */
fread(&(ext2_block.s_feature_ro_compat), 4, 1, fp);
```

```
/* readonly-compatible feature set */
/*68*/fread(&(ext2_block.s_uuid), 1, 16, fp);

/* 128-bit uuid for volume */
fread(&(ext2_block.s_volume_name), 1, 16, fp);

/* volume name */
fread(&(ext2_block.s_last_mounted), 1, 64, fp);

/* directory where last mounted */
fread(&(ext2_block.s_algorithm_usage_bitmap), 4, 1, fp);
fread(&(ext2_block.s_prealloc_blocks), 1, 1, fp);
fread(&(ext2_block.s_prealloc_dir_blocks), 1, 1, fp);
fread(&(ext2_block.s_padding1), 2, 1, fp);
fread(&(ext2_block.s_journal_uuid), 1, 16, fp);

/* uuid of journal superblock */
fread(&(ext2_block.s_journal_inum), 4, 1, fp);

/* 日志文件的 inode 号数 */
fread(&(ext2_block.s_journal_dev), 4, 1, fp);

/* 日志文件的 设备 号 */
fread(&(ext2_block.s_last_orphan), 4, 1, fp);

/* start of list of inodes to delete */;

printf("%20s  :  %-lu\n", "s_inodes_count", ext2_block
        .s_inodes_count);
printf("%20s  :  %-lu\n", "s_blocks_count", ext2_block
        .s_blocks_count);
```

```
printf("%20s  :  %-lu\n", "s_r_blocks_count", ext2_block
        .s_r_blocks_count);
printf("%20s  :  %-lu\n", "s_free_blocks_count", ext2_block
        .s_free_blocks_count);
printf("%20s  :  %-lu\n", "s_free_inodes_count", ext2_block
        .s_free_inodes_count);
printf("%20s  :  %-lu\n", "s_first_data_block", ext2_block
        .s_first_data_block);
printf("%20s  :  %-lu\n", "s_log_block_size", ext2_block
        .s_log_block_size);
printf("%20s  :  %-lu\n", "s_blocks_per_group", ext2_block
        .s_blocks_per_group);
printf("%20s  :  %-lu\n", "s_mtime", ext2_block.s_mtime);
printf("%20s  :  %-lu\n", "s_wtime", ext2_block.s_wtime);
printf("%20s   :  %-d\n", "s_mnt_count", ext2_block
        .s_mnt_count);
printf("%20s  :  %-x\n", "s_magic", ext2_block.s_magic);

int G = (ext2_block.s_blocks_count - ext2_block
        .s_first_data_block - 1) / ext2_block
        .s_blocks_per_group + 1;
printf("%20s  :  %-d\n", "block group", G);
printf("%20s  :  %-lu\n", "s_first_ino", ext2_block
        .s_first_ino);
printf("%20s  :  %-u\n", "s_inode_size", ext2_block
        .s_inode_size);
//////////////////////////////////////////////////////////////////
fseek(fp, 1024 * 2, SEEK_SET);
fread(&(g_block_bitmap), 4, 1, fp);
fread(&(g_inode_bitmap), 4, 1, fp);
fread(&(g_inode_table), 4, 1, fp);
```

```
    printf("%20s  :  %-lu\n", "g_block_bitmap", g_block_bitmap);
    printf("%20s  :  %-lu\n", "g_inode_bitmap", g_inode_bitmap);
    printf("%20s  :  %-lu\n", "g_inode_table", g_inode_table);
    ////////////////////////////////////////////////////////////////
    /// inodetable is arranged by inode ascendingly
    /// root_inode  = 2;
    /// now find root in the table
    fseek(fp, 1024 * g_inode_table + ext2_block.s_inode_size, SEEK_SET);
    fread(&(i_mode), 2, 1, fp);
    printf("%20s  :  %-x\n", "i_mode", i_mode);
    fread(&(i_mode), 2, 1, fp);
    fread(&(i_size), 4, 1, fp);
    printf("%20s  :  %-lu\n", "i_size", i_size);
    fseek(fp, 20, SEEK_CUR);
    fread(&(i_blocks), 4, 1, fp);
    printf("%20s  :  %-lu\n", "i_blocks", i_blocks);
    fseek(fp, 8, SEEK_CUR);
    fread(&(i_block), 4, 1, fp);
    printf("%20s  :  %-lu\n", "i_block", i_block);
    /// now find root dir file
    fseek(fp, 1024 * i_block, SEEK_SET);
    /// entry list
    printf("\t***  root dir file details ***\n");
    while (1)
    {
        printf("-------------------------------------------------\n");
        fread(&(inode), 4, 1, fp);
        if (0 == inode)
            break;
        printf("%10s  :  %-lu\t", "inode", inode);
        fread(&(rec_len), 2, 1, fp);
```

```
        printf("%10s  :  %-u\n", "rec_len", rec_len);
        fread(&(name_len), 1, 1, fp);
        printf("%10s  :  %-u\t", "name_len", name_len);
        fread(&(file_type), 1, 1, fp);
        if (1 == file_type)
            regularnode = inode;
        printf("%10s  :  %-u\n", "file_type", file_type);
        printf("%10s  :  ", "file_name");
        for (i = 0;

                i < name_len;

                i++)
        {
            fread(&(name), 1, 1, fp);
            putchar(name);
        }
        printf("\n");
        fseek(fp, rec_len - (8 + name_len), SEEK_CUR);
    }
    /// regular file ///
    if (regularnode < ext2_block.s_inodes_per_group && regularnode)
    {
        printf("\t*** regular file at root dir ***\n");
        fseek(fp, 1024 * g_inode_table + (regularnode - 1) * ext2_block
                .s_inode_size, SEEK_SET);
        fread(&(i_mode), 2, 1, fp);
        printf("%20s  :  %-x\n", "i_mode", i_mode);
        fread(&(i_mode), 2, 1, fp);
        fread(&(i_size), 4, 1, fp);
        printf("%20s  :  %-lu\n", "i_size", i_size);
```

```
        fseek(fp, 20, SEEK_CUR);
        fread(&(i_blocks), 4, 1, fp);
        printf("%20s  :  %-lu\n", "i_blocks", i_blocks);
        fseek(fp, 8, SEEK_CUR);
        fread(&(i_block), 4, 1, fp);
        printf("%20s  :  %-lu\n", "i_block", i_block);
        /// now find file
        fseek(fp, 1024 * i_block, SEEK_SET);
        for (i = 0;

                i < i_size;

                i++)
        {
            fread(&(name), 1, 1, fp);
            putchar(name);
        }
        printf("-------------------------------------------------\n");
    }
    /// close file
    fclose(fp);
}

int main(int argc, char** argv)
{
    Initial(argc, argv);
    return 0;
}
```