哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2021 秋季

课程名称：操作系统

实验名称：基于 FUSE 的青春版 EXT2 文件系统

学生班级：19 级计科 10 班

学生学号：190111026

学生姓名：郭毅安

评阅教师：

报告成绩：

实验与创新实践教育中心制

2020 年 9 月

一、实验详细设计

1、 总体设计方案

分析：

文件系统是操作系统用于明确存储设备（磁盘）或分区上的文件（包括文件、目录、快捷方式等）的方法和数据结构，即 在特定存储设备上组织文件的方法。简单地说，文件系统就是在特定存储设备上组织文件的方法。

本实验以 Linux 系统中的 EXT2 文件系统为例，熟悉该文件系统内部数据结构的组织方式和基本处理流程，并基于 FUSE 设计并实现一个可以真正在 Linux 上跑 的文件系统。

总体设计：

参考给出的已经实现的 FUSE 文件系统 simplefs，并增添数据位图的实现。完成介质数据结构，实现部分工具函数和钩子函数并通过脚本测试

2、 功能详细说明

关键数据结构：

```
struct newfs_inode {     //文件系统的索引项
    uint32_t             ino;                        // inode 位图里的下标
    int                  size;                       /* 文件已占用空间 */
    int                  dir_cnt;
    struct newfs_dentry* dentry;                     /* 指向该inode的dentry */
    struct newfs_dentry* dentrys;                    /* 所有目录项 */
    uint8_t*             data;
};

struct newfs_dentry {    //文件系统的目录项
    char                 fname[NEWFS_MAX_FILE_NAME];
    uint32_t             ino;                          // 其所指向的 ino 号
    struct newfs_inode*  inode;                        /* 指向 inode */
    NEWFS_FILE_TYPE      ftype;
    struct newfs_dentry* brother;                      /* 兄弟 */
    struct newfs_dentry* parent;
};

struct newfs_super {   //文件系统的超级快
    uint32_t             magic;
    int                  driver_fd;
    int                  sz_io;                  //size
    int                  sz_disk;
    int                  sz_usage;

    int                  max_ino;
```

```
    uint8_t*            map_inode;              //inode 位图
    int                 map_inode_blks;         //inode 位图占用的块数
    int                 map_inode_offset;       //inode 位图在磁盘上的偏移
    uint8_t*            map_data;               //data 位图
    int                 map_data_blks;          //data 位图占用的块数
    int                 map_data_offset;        //data 位图在磁盘上的偏移

    int                 inode_offset;
    int                 data_offset;

    boolean             is_mounted;

    struct newfs_dentry* root_dentry;            //根目录 root
};

struct newfs_super_d {     //磁盘结构的超级快
    uint32_t           magic_num;                    // 幻数
    int                max_ino;                      // 最多支持的文件数
    int                map_inode_blks;               // inode 位图占用的块数
    int                map_inode_offset;             // inode 位图在磁盘上的偏
移
    int                map_data_blks;                // data 位图占用的块数
    int                map_data_offset;              // data 位图在磁盘上的偏移
    int                sz_usage;

    int                inode_offset;
    int                data_offset;
};

struct newfs_inode_d {     //磁盘结构的索引块
    int                 ino;                       // 在 inode 位图中的下标
    int                 size;                      // 文件已占用空间
    int                 link;                      // 链接数
    NEWFS_FILE_TYPE     ftype;                     // 文件类型（目录类型、普通文件
类型）
    int                 dir_cnt;                   // 如果是目录类型文件，下面有几
个目录项
    int                 block_pointer[6];          // 数据块指针（可固定分配）
};

struct newfs_dentry_d {    //磁盘结构的目录项
    char               fname[NEWFS_MAX_FILE_NAME];   // 指向的 ino 文件名
    NEWFS_FILE_TYPE    ftype;                        // 指向的 ino 文件类型
    int                 ino;                         // 指向的 ino 号
```

```
    int                 valid;                          // 该目录项是否有效
};
```

功能点代码和流程:

（1） mount（挂载）

首先初始化超级块根目录，然后完成驱动读。接着判断幻数，若不相等则估算各部分大小，再进行 layout 布局。然后建立 in-memory 结构，再读索引位图和数据位图。最后如果当前磁盘中有系统，系统没损坏，且是能识别的文件系统，则分配根节点，然后完成超级块的实现。

```
    newfs_super.sz_usage   = newfs_super_d.sz_usage;      /* 建立 in-memory 结构 */
    newfs_super.max_ino = newfs_super_d.max_ino;
    newfs_super.map_inode_blks = newfs_super_d.map_inode_blks;
    newfs_super.map_inode_offset = newfs_super_d.map_inode_offset;

    newfs_super.map_data_blks = newfs_super_d.map_data_blks;
    newfs_super.map_data_offset = newfs_super_d.map_data_offset;

    newfs_super.inode_offset = newfs_super_d.inode_offset;
    newfs_super.data_offset = newfs_super_d.data_offset;

    newfs_super.map_inode = (uint8_t *)malloc(NEWFS_BLKS_SZ(newfs_super_d.map_inode_blks));
    newfs_super.map_data = (uint8_t *)malloc(NEWFS_BLKS_SZ(newfs_super_d.map_data_blks));

    //读索引位图
    if (newfs_driver_read(newfs_super_d.map_inode_offset, (uint8_t *)(newfs_super.map_inode),
                          NEWFS_BLKS_SZ(newfs_super_d.map_inode_blks)) != NEWFS_ERROR_NONE) {
                                                          //读 newfs_super_d.map_inode_blks
        return -NEWFS_ERROR_IO;
    }
    //读数据位图
    if (newfs_driver_read(newfs_super_d.map_data_offset, (uint8_t *)(newfs_super.map_data),
                          NEWFS_BLKS_SZ(newfs_super_d.map_data_blks)) != NEWFS_ERROR_NONE) {
                                                          //读 newfs_super_d.map_inode_blks
        return -NEWFS_ERROR_IO;
    }
```

```
    if (is_init) {                                  /* 分配根节点 */
        root_inode = newfs_alloc_inode(root_dentry);
        newfs_sync_inode(root_inode);
    }
    root_inode            = newfs_read_inode(root_dentry, NEWFS_ROOT_INO);
    root_dentry->inode    = root_inode;
    newfs_super.root_dentry = root_dentry;
    newfs_super.is_mounted  = TRUE;
```

（2） umount（卸载）

处理 super 块，写回磁盘，再写超级块、索引位图和数据位图，最后释放位图在内存中的空间

```
newfs_sync_inode(newfs_super.root_dentry->inode);     /* 从根节点向下刷写节点 */
    newfs_super_d.magic_num           = NEWFS_MAGIC_NUM;
    newfs_super_d.map_inode_blks      = newfs_super.map_inode_blks;
    newfs_super_d.map_inode_offset    = newfs_super.map_inode_offset;
    newfs_super_d.map_data_blks       = newfs_super.map_data_blks;
    newfs_super_d.map_data_offset     = newfs_super.map_data_offset;
    newfs_super_d.inode_offset        = newfs_super.inode_offset;
    newfs_super_d.data_offset         = newfs_super.data_offset;
    newfs_super_d.sz_usage            = newfs_super.sz_usage;

    if (newfs_driver_write(NEWFS_SUPER_OFS, (uint8_t *)&newfs_super_d,
                           sizeof(struct newfs_super_d)) != NEWFS_ERROR_NONE) {
        return -NEWFS_ERROR_IO;
    }

    if (newfs_driver_write(newfs_super_d.map_inode_offset, (uint8_t *)(newfs_super.map_inode),
                           NEWFS_BLKS_SZ(newfs_super_d.map_inode_blks)) != NEWFS_ERROR_NONE) {
        return -NEWFS_ERROR_IO;
    }

    if (newfs_driver_write(newfs_super_d.map_data_offset, (uint8_t *)(newfs_super.map_data),
                           NEWFS_BLKS_SZ(newfs_super_d.map_data_blks)) != NEWFS_ERROR_NONE) {
        return -NEWFS_ERROR_IO;
```

```
    }


    free(newfs_super.map_inode);
    free(newfs_super.map_data);
    ddriver_close(NEWFS_DRIVER());
```

（3） mkdir

确认是文件后，新建目录项，再保存上级目录，然后为新目录分配一个 inode，并占用对应的位图，最后把这个新建的目录项链接在上一级 inode 下。

```
    fname  = newfs_get_fname(path);
    dentry = new_dentry(fname, NEWFS_DIR);
    dentry->parent = last_dentry;
    inode  = newfs_alloc_inode(dentry);
    newfs_alloc_dentry(last_dentry->inode, dentry);
```

（4） touch

先确认待创建文件不存在，再创建新 dentry，并为止创建新 inode，最后添加 dentry 到上级 inode 的列表中，成为新的一个 inode。

```
    dentry->parent = last_dentry;
    inode = newfs_alloc_inode(dentry);
    newfs_alloc_dentry(last_dentry->inode, dentry);
```

（5） ls

找到文件后，依次读取目录项，将 fname 放入 buf 中，并使目录项偏移一位，实现访问下目录项，若不为空时则一直访问，并输出。

```
if (is_find) {
        inode = dentry->inode;
        sub_dentry = newfs_get_dentry(inode, cur_dir);
        if (sub_dentry) {
            filler(buf, sub_dentry->fname, NULL, ++offset);
        }
        return NEWFS_ERROR_NONE;
    }
```

3、 *实验特色*

（1） 实现了索引位图和数据位图。

（2） 操作简单

测试结果：

```
190111026@OSLabExecNode0:~/user-land-filesystem/fs/newfs0$ ./tests/fs_test.sh
目标设备 /home/guests/190111026/ddriver
8192+0 records in
8192+0 records out
4194304 bytes (4.2 MB, 4.0 MiB) copied, 0.0531293 s, 78.9 MB/s
>>>>>>>>>>>>>>>>>>> TEST_MOUNT
pass: [all-the-mount-test]
<<<<<<<<<<<<<<<<<<<

>>>>>>>>>>>>>>>>>>> TEST_MKDIR
TEST: mkdir ./mnt/dir0
pass: -> mkdir ./mnt/dir0
TEST: mkdir ./mnt/dir0/dir0
pass: -> mkdir ./mnt/dir0/dir0
TEST: mkdir ./mnt/dir0/dir0/dir0
pass: -> mkdir ./mnt/dir0/dir0/dir0
TEST: mkdir ./mnt/dir1
pass: -> mkdir ./mnt/dir1
<<<<<<<<<<<<<<<<<<<
```

```
>>>>>>>>>>>>>>>>>>> TEST_TOUCH
TEST: touch ./mnt/file0
pass: -> touch ./mnt/file0
TEST: touch ./mnt/dir0/file0
pass: -> touch ./mnt/dir0/file0
TEST: touch ./mnt/dir0/dir0/file0
pass: -> touch ./mnt/dir0/dir0/file0
TEST: touch ./mnt/dir0/dir0/dir0/file0
pass: -> touch ./mnt/dir0/dir0/dir0/file0
TEST: touch ./mnt/dir1/file0
pass: -> touch ./mnt/dir1/file0
<<<<<<<<<<<<<<<<<<<

>>>>>>>>>>>>>>>>>>> TEST_LS
TEST: ls ./mnt/
dir0  dir1  file0
pass: -> ls ./mnt/
TEST: ls ./mnt/dir0
dir0  file0
pass: -> ls ./mnt/dir0
TEST: ls ./mnt/dir0/dir0
dir0  file0
pass: -> ls ./mnt/dir0/dir0
TEST: ls ./mnt/dir0/dir0/dir0
file0
pass: -> ls ./mnt/dir0/dir0/dir0
TEST: ls ./mnt/dir1
file0
pass: -> ls ./mnt/dir1
<<<<<<<<<<<<<<<<<<<
```

```
>>>>>>>>>>>>>>>>>>>> TEST_REMOUNT
pass: -> fusermount -u ./mnt
pass: -> ../build/newfs --device=/home/guests/190111026/ddriver ./mnt
TEST: ls ./mnt/
dir0  dir1  file0
pass: -> ls ./mnt/
TEST: ls ./mnt/dir0
dir0  file0
pass: -> ls ./mnt/dir0
TEST: ls ./mnt/dir0/dir0
dir0  file0
pass: -> ls ./mnt/dir0/dir0
TEST: ls ./mnt/dir0/dir0/dir0
file0
pass: -> ls ./mnt/dir0/dir0/dir0
TEST: ls ./mnt/dir1
file0
pass: -> ls ./mnt/dir1
pass: -> fusermount -u ./mnt
<<<<<<<<<<<<<<<<<<<<

pass: 恭喜你，通过所有测试 (23/23)
```

二、用户手册

实现的文件系统中的所有命令使用方式

挂载：

输入指令：./build/newfs --device=ddriver 的路径 路径

例：./build/newfs --device=/home/guests/190111026/ddriver -f -d -s ./tests/mnt/

创建文件夹：mkdir + 目录名

创建文件：touch + 文件名

查看目录项：ls

卸载： fusermount -u + 路径

三、实验收获和建议

（1） 收获：

加深了对进程、锁、页表、文件系统等概念和知识的理解和掌握，将操作系统这门课上学到的理论知识较好的运用于实践中，提高了自己的代码和调试能力，积累了 linux 环境下实现代码和程序的经验。

（2） 感受：

操作系统是一门很深奥的学科，xv6 虽小，但其玄妙和深奥仍然让人感叹，好在有各位老师和助教们的悉心指导，让我顺利完成这一系列实验，在此表示衷心感谢。

（3） 问题建议：

建议延长实验间隔，因为一个学期不可能只有操作系统实验这一门实验课，建议 ddl 间隔定为 10 天或两星期。

四、参考资料

实验指导书
Simplefs 文件系统