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

实验报告

开课学期：2024 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：计科四班

学生学号：220110415

学生姓名：黄宇浩

评阅教师：

报告成绩：

实验与创新实践教育中心制

2024 年 9 月

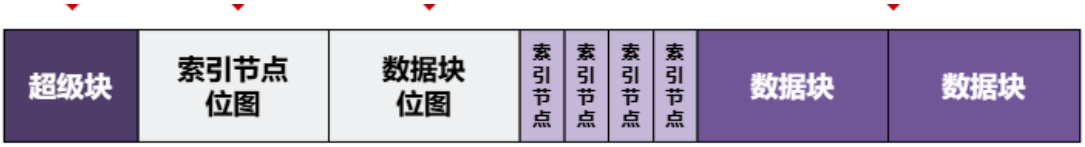
一、实验详细设计

图文并茂地描述实验实现的所有功能和详细的设计方案及实验过程中的特色部分。

1、 总体设计方案

详细阐述文件系统的总体设计思路，包括系统架构图和关键组件的说明。

系统架构图和实验指导书一致，分为五个部分：超级块、索引位图、数据位图、索引节点区和数据区



其中 Layout 分布为：超级块索引位图和数据块位图均占用一个逻辑块，索引节点区一个索引节点占用 1 个逻辑块。一个文件由一个索引节点和 6 个数据块构成，一个数据块一个索引节点。因此索引节点区的大小为 585 个逻辑块（至多有 585 个文件），数据区的大小为 3508 个逻辑块。

一个逻辑块大小 1024B，IO 块大小 512B，磁盘大小 4MB。

2、 功能详细说明

每个功能点的详细说明（关键的数据结构、核心代码、流程等）

一.数据结构定义

这里只阐述和参考代码不同的定义点

1. 超级块里维护了索引位图、数据位图、索引节点区和数据区的偏移量（offset）和占用逻辑块的大小（blks）。Super_d 里也包含了这些属性
2. Inode 数据结构里添加了 data_blks_cnt 属性，用于指示使用了几个数据块，方便进行数据位图的维护。Inode_d 里同样设置了这个属性
3. Entry 和 Entry_d 的属性与参考代码一致

二，必做函数实现

1. newfs_init：挂载函数。流程是：首先初始化一些基础属性（如 sz_io），然后从磁盘里读取 super_d，根据幻数判断是否为第一次挂载。若是第一次则初始化 super_d 的属性（如 map_inode_offset 的设置等），然后把 super_d 的属性赋给 super。这样就完成了 super 属性的设置。然后从磁盘读取索引节点位图和数据块位图。最后是根节点的设置，如果是第一次挂载就分配根节点并写回磁盘，否则从磁盘里获取根节点。
2. newfs_destroy：卸载函数。首先把索引节点区和数据区的数据写回磁盘，然后通过 super 的属性构建 super_d 并把 super_d 写回磁盘，最后把索引节点位图和数据位图写回磁盘并释放掉
3. newfs_mkdir & newfs_mknod：创建文件夹和文件。首先找到父目录项 father_dentry。然后新建目录项（根据属性）和索引节点，并把新建的目录项与父目录项对应的索引节点关联在一起。
4. newfs_getattr：首先找到该路径文件的索引节点，然后根据索引节点中的信息填充状态并返回，填充方式与参考代码一致
5. newfs_readdir：即找到该路径文件的索引节点，并找到其子 dentry 里的第 offset 个目录

项，用 filler 填充该目录项信息到 buf。

三，选做函数实现：

1. newfs_write & newfs_read：找到该路径文件的索引节点，并填充该索引节点的数据区(从数据区读取数据)。核心点在于判断写/读的区域的上界不能超过文件的最大大小。同时还要修改数据位图（动态分配）

```
if(offset + size > BLKS_SZ(super.file_max)){
    size_ret = BLKS_SZ(super.file_max) - offset;
}
```

2. newfs_unlink：主要使用了两个工具函数：drop_inode 用于递归地删除该索引节点的数据（文件）或子目录项（文件夹）和 drop_dentry 用于切断父 inode 与待删除 dentry 之间的联系
3. newfs_rename：为 to 新建文件，删除 to 原有的 inode，并把 to 的 inode 改成 from 的 inode，最后删除 from
4. newfs_truncate：改变文件的使用大小并修改数据块位图（说实话这个函数有点奇怪，为什么要修改文件的使用大小？）
5. newfs_access：判断文件的可访问性。根据文件的存在性和权限判断。

四，辅助函数实现：

1. read_from_disk & write_from_disk：驱动读和驱动写。根据 offset 和 size 找到覆盖读写范围的、整数块的 IO 块。read 是先从磁盘里读取数据填充 temp，再从 temp 中截取需要的数据返回。Write 是先从磁盘中读出 temp，拷贝要写的内容到 temp 里，再把 temp 写回磁盘。
2. read_inode_from_disk：根据 ino 从磁盘中获取并构建 inode。根据 ino 从磁盘中读出 inode_d，根据 inode_d 构建 inode，把 inode 和 dentry 关联上。然后根据 inode 的类型：如果是目录文件，则从数据区读出所有的 dentry_d，根据 dentry_d 构建 dentry_cur，并把 dentry_cur 作为 inode 的子 dentry 与 inode 关联起来。如果是普通文件，则从磁盘中读取数据并填充索引节点的数据区
3. insert_dentry_to_inode：将 dentry 作为 inode 的子目录之一与 inode 进行关联,采用头插法。和参考代码有不同的是：添加了 inode->size += sizeof(struct newfs_dentry_d);修改目录文件大小
4. write_inode_to_disk：把 inode 写回磁盘。根据 inode 构建 inode_d，将 inode_d 写回磁盘。如果是目录，则递归把所有子目录项对应的 inode 也写回磁盘（调用此函数）。如果是文件，则把数据写回磁盘。
5. alloc_free_inode：从索引节点位图中找一个没用过的 inode，初始化相关数据并与 dentry 关联上。
6. cal_path_level：计算路径的层级（按‘/’划分，只有根目录的话层级为 0）
7. lookup：根据路径寻找 path 对应的 dentry，如果找到返回 dentry，找不到父目录的 dentry（上一个有效路径）。从根目录找起，递归检索目录项的名字是否与路径中相应层级的名字一致
8. get_fname：获取文件名(从路径的末位获取（最后一个‘/’后面）)
9. get_ith_son_dentry：查找并返回 inode 的第 dir 个子 dentry
10. modify_data_map：自建函数，用于当 inode 的数据改变（一般是增加时）时修改数据块位图。首先若是目录文件，判断其 dir_cnt 是否超过了数据块数*每页最大数据块数（通过每个逻辑块大小/dentry_d 的大小得到），超过了就分配新的数据块并修改数据块位图。若是普通文件就根据文件使用大小重新分配数据块数并修改数据块位图。

```
else if(IS_FILE(inode)){
    inode->data_blks_cnt = FIND_UP_EDGE(inode->size,BLOCK_SIZE()) / BLOCK_SIZE();
    for(int i=0;i<super.file_max;i++,data_ino++){
        if(i < inode->data_blks_cnt){
            super.map_data[data_ino / BITS_IN_A_BYTE] |= (0x1 << (data_ino % BITS_IN_A_BYTE));
        }
        else{
            super.map_data[data_ino / BITS_IN_A_BYTE] &= (uint8_t)(~(0x1 << (data_ino % BITS_IN_A_BYTE)));
        }
    }
}
```

11. clear_data_map：根据 ino 清空数据块位图
12. drop_inode：删除 inode，若是目录项就迭代删除子目录项，若是文件就清空数据。同时修改数据位图和索引位图。最后用 free 释放
13. drop_dentry： 删除关联，切断 inode 与子 dentry 之间的联系。主要是修改 inode(其它子 dentry 和一些属性)

五，宏定义

与参考代码有较多重合，不过每个宏根据其在文件系统中的作用进行了重命名

如：

```
#define MKDIR_ON_FILE_ERROR ENXIO
#define MKNODE_ON_FILE_ERROR ENXIO
```

还有自定义的宏：

```
#define MAX_DENTRY_D_PER_BLOCK (BLOCK_SIZE() / sizeof(struct newfs_dentry_d))
```

至此，就完成了文件系统的挂载与卸载、ls、mkdir、touch、rm、rm -r、文件的读取与修改、mv 全部实验功能的实现。

```
pass: case 7.2 - copy /home/students/220
=========================================
Score: 34/34
pass: 恭喜你，通过所有测试 (34/34)
```

3、 实验特色

实验中你认为自己实现的比较有特色的部分，包括设计思路、实现方法和预期效果。

我认为比较有特色的部分在于位图的修改。在参考代码中对索引位图的修改，其的方法是先按字节遍历，内部再按比特寻址，直到位数等于 ino。

但我思考后觉得，既然 ino 就是位图中的第 i 位，那么寻找字节直接用 ino/8 即可，然后内部的比特数就是 ino%8，这样就可以避免重复性的遍历，一步到位。（不过对于数据块位图，起始偏移是 ino*6）

因此，在数据块位图的修改中，我就采用了这种方法

以这段代码为例

```
int data_ino = inode->ino * DATA_PART_PER_FILE;
if(IS_DIR(inode)){
    data_ino = data_ino + inode->data_blks_cnt - 1;
    while(inode->dir_cnt > inode->data_blks_cnt * MAX_DENTRY_D_PER_BLOCK){
        inode->data_blks_cnt++;
        data_ino++;
        super.map_data[data_ino / BITS_IN_A_BYTE] |= (0x1 << (data_ino % BITS_IN_A_BYTE));
    }
```

首先 data_ino 定位到了 ino 对应的数据块位图的起始位置，然后根据要修改的 inode 的数据块数，偏移 data_ino，并直接修改数据块位图，这里 BITS_IN_A_BYTE 就是 8.

实操检验：

```
220110415@comp3:~/user-land-filesystem/fs/newfs/tests$ touch ./mnt/file0
220110415@comp3:~/user-land-filesystem/fs/newfs/tests$ cd ..
220110415@comp3:~/user-land-filesystem/fs/newfs$ fusermount -u ./tests/mnt
```

这里我进行了挂载、创建文件 file0 并卸载的操作

然后查看磁盘信息（2048 定位到 0x800）：

```
000007F0  00 00 00
00000800  41 00 00
00000810  00 00 00
```

可以看到，第一个字节的数据是 41，即 0100 0001，第一个比特和第七个比特被修改，说明数据块位图修改正确，方法成立。

二、遇到的问题及解决方法

*列出实验过程中遇到的主要问题，包括技术难题、设计挑战等。对应每个问题，提供采取的**解决策略**，以及解决问题后的效果评估。*

1.实验整体架构梳理

首先就是反复阅读实验指导书，厘清每个数据结构的含义。比如 inode 与 dentry 是一一对应的，每个 inode 还维护了子 dentry 序列（采用链表的形式组织）

2.测评程序无法编译的问题：用 printf 打印了每个区域的大小，发现数据区大小的代码写错了，修改。就能成功运行测评脚本了。

3.索引位图不正确：这个问题折磨了很久，在 write_to_disk、write_inode_to_disk 函数中都添加了很多的打印信息，比如打印出索引位图第一个字节的内容等等，同时也去查看了磁盘里相应位置的数据，发现明明传入 1，但是显示的是 A0，后来才发现是在

```
if(write_to_disk(super.map_inode_offset,(uint8_t*)super.map_inode,BLKS_SZ(super.map_inode_blks)) != NO_ERROR){
    return;
}
```

里，对 super.mao_inode 添加了&，打印出来的是索引位图的地址，修改后才解决了索引位图的问题以及 remount 时数据丢失的问题。

4.一开始建了一个 tools.c 文件存放所有工具函数，但是后来删了这个文件后却无法编译成功了。解决办法是清空了 build 文件夹中的内容并重新用 cmake ..命令构建

5.其它的小问题：如数据位图不正确，发现一开始根本没有修改数据位图，修改了就好了。其它没什么特别大的问题了。

三、实验收获和建议

实验中的收获、感受、问题、建议等。

收获：对文件系统更熟悉了，包括 super 块、inode 和 dentry 等数据结构，也对各种文件指令内部的逻辑更加清楚了。同时也增加了 C 语言知识，比如规范的宏定义和枚举类型

感受：主要感受就是代码量太大了，总共写了两天半的代码，超过 1000 行。不过只要理解好了原理就不算难，毕竟大部分跟着参考代码做就行，只有少量的自己设计的部分（比如数据块位图部分）

建议：可以帮忙实现一些简单的工具函数的模板(比如 get_fname 之类的)，减少代码量。要不然代码量实在是大。

总的来说，还是很多收获的一次实验，实验指导书写的很详细，问题指引也很全面，赞！

四、参考资料

实验过程中查找的信息和资料

1. 实验指导书
2. Simplefs 参考代码
3. Microsoft Copilot AI 助手