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

实验报告

开课学期：2023 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：计算机 6 班

学生学号：210110612

学生姓名：章懿

评阅教师：

报告成绩：

实验与创新实践教育中心制

2023 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

对实现整个文件系统的分析说明

文件系统结构：

- 超级块（Superblock）：存储文件系统的整体信息，如块大小、总块数、空闲块数等。
- 索引节点位图（Inode Bitmap）：表示索引节点的占用情况。
- 数据块位图（Data Bitmap）：表示数据块的占用情况。
- 索引节点区（Inode Table）：存储文件和目录的元数据。
- 数据块区（Data Blocks）：存储文件和目录的实际数据。

如下图：

```
22    | BSIZE = 1024 B |
23    | Super(1) | Inode Map(1) | DATA Map(1) | INODE(240) | DATA(*) |
```

I/O 交互：

通过封装的 newfs_driver_read 和 newfs_driver_write 函数与磁盘进行读写操作。

文件系统接口：

实现了 FUSE（文件系统用户空间）接口，如 init、destroy、mkdir、getattr 等，以支持文件系统的基本操作。

2、 功能详细说明

每个功能点的详细说明（关键的数据结构、代码、流程等）

1. 挂载文件系统（newfs_init）

先读取磁盘中的磁盘大小，IO 大小和逻辑块大小到 in-Mem 的超级块中。

```
super.fd = ddriver_open(newfs_options.device);
if (super.fd < 0)
{
    /* code */
    return super.fd;
}

ddriver_ioctl(NEWFS_DRIVER(), IOC_REQ_DEVICE_SIZE, &super.sz_disk);
ddriver_ioctl(NEWFS_DRIVER(), IOC_REQ_DEVICE_IO_SZ, &super.sz_io);
super.sz_blk = super.sz_io * 2;
```

然后判断是否已初始化，若没有初始化，则需要重新估算磁盘布局信息：

```
if(super_d.magic != NEWFS_MAGIC_NUM) { /* 幻数无 */
    // 重新估算磁盘布局信息
    super_d.sb_offset = 0;
    super_d.sb_blks = 1;

    super_d.ino_map_offset = super_d.sb_offset + NEWFS_BLKS_SZ(super_d.sb_blks);
    super_d.ino_map_blks = 1;

    super_d.data_map_offset = super_d.ino_map_offset + NEWFS_BLKS_SZ(super_d.ino_map_blks);
    super_d.data_map_blks = 1;

    super_d.ino_offset = super_d.data_map_offset + NEWFS_BLKS_SZ(super_d.data_map_blks);
    super_d.ino_blks = super.sz_disk / ((NEWFS_INODE_PER_FILE + NEWFS_DATA_PER_FILE) * NEWFS_BLK_SZ());

    super_d.data_offset = super_d.ino_offset + NEWFS_BLKS_SZ(super_d.ino_blks);
    super_d.data_blks = super.sz_disk / super.sz_blk - 1 - 1 - 1 - super_d.ino_blks;

    super_d.ino_max = super_d.ino_blks - super_d.sb_blks - super_d.ino_map_blks - super_d.data_map_blks;
    super_d.file_max = NEWFS_DATA_PER_FILE * NEWFS_BLK_SZ();

    super_d.sz_usage = 0;
    is_init = TRUE;
}
```

若已经初始化，则直接读取，填充磁盘布局信息

```
/* 建立 in-memory 结构 */
// 填充磁盘布局信息
super.sz_usage = super_d.sz_usage;

super.sb_blks = super_d.sb_blks;
super.sb_offset = super_d.sb_offset;

super.ino_map_offset = super_d.ino_map_offset;
super.ino_map_blks = super_d.ino_map_blks;

super.data_map_blks = super_d.data_map_blks;
super.data_map_offset = super_d.data_map_offset;

super.ino_offset = super_d.ino_offset;
super.ino_blks = super_d.ino_blks;

super.data_offset = super_d.data_offset;
super.data_blks = super_d.data_blks;

super.ino_max = super_d.ino_max;
super.file_max = super_d.file_max;

super.map_inode = (uint8_t *)malloc(NEWFS_BLKS_SZ(super_d.ino_map_blks));
super.map_data = (uint8_t *)malloc(NEWFS_BLKS_SZ(super_d.data_map_blks));
```

读取索引节点，数据块位图，建立根目录节点，生成层级。

```
// 读取索引节点
if (newfs_driver_read(super_d.ino_map_offset, (uint8_t *)(super.map_inode),
                      NEWFS_BLKS_SZ(super_d.ino_map_blks)) != NEWFS_ERROR_NONE) {
    return -NEWFS_ERROR_IO;
}
// 读取数据块节点
if (newfs_driver_read(super_d.data_map_offset, (uint8_t *)(super.map_data),
                      NEWFS_BLKS_SZ(super_d.data_map_blks)) != NEWFS_ERROR_NONE) {
    return -NEWFS_ERROR_IO;
}

/* 分配根节点 */
// 创建空根目录和节点
if (is_init) {
    root_inode = newfs_alloc_inode(root_dentry);

    root_dentry->inode = root_inode;
    // 给根节点分配第一个数据块
   if(newfs_alloc_datablk(root_dentry) < 0)
       printf("新增数据块失败");

    newfs_sync_inode(root_inode);
}

/* 读取根目录inode，生成层级 */
root_inode            = newfs_read_inode(root_dentry, NEWFS_ROOT_INO);
root_dentry->inode    = root_inode;
super.root_dentry = root_dentry;
super.is_mounted  = TRUE;
```

设计关键：确保超级块数据准确读取，正确初始化根目录。

2. 卸载文件系统（newfs_destroy）

首先将 in-Mem 的超级块的数据写入到 to-Disk 的超级块中：

```
super_d.magic = NEWFS_MAGIC_NUM;

super_d.sz_usage = super.sz_usage;

super_d.sb_offset = super.sb_offset;
super_d.sb_blks = super.sb_blks;

super_d.ino_map_blks = super.ino_map_blks;
super_d.ino_map_offset = super.ino_map_offset;

super_d.data_map_offset = super.data_map_offset;
super_d.data_map_blks = super.data_map_blks;

super_d.ino_offset = super.ino_offset;
super_d.ino_blks = super.ino_blks;

super_d.data_blks = super.data_blks;
super_d.data_offset = super.data_offset;

super_d.ino_max = super.ino_max;
super_d.file_max = super.file_max;
```

然后使用 newfs_driver_write 钩子将超级块数据、索引位图数据、数据块位图数据写回：

```
// 写回超级块
if (newfs_driver_write(NEWFS_SUPER_OFS, (uint8_t *)&super_d,
                       sizeof(struct newfs_super_d)) != NEWFS_ERROR_NONE) {
    return -NEWFS_ERROR_IO;
}
// 写回索引位图
if (newfs_driver_write(super_d.ino_map_offset, (uint8_t *)(super.map_inode),
                       NEWFS_BLKS_SZ(super_d.ino_map_blks)) != NEWFS_ERROR_NONE) {
    return -NEWFS_ERROR_IO;
}
// 写回数据位图
if (newfs_driver_write(super_d.data_map_offset, (uint8_t *)(super.map_data),
                       NEWFS_BLKS_SZ(super_d.data_map_blks)) != NEWFS_ERROR_NONE) {
    return -NEWFS_ERROR_IO;
}
```

并且从根节点向下刷写节点：

```
newfs_sync_inode(super.root_dentry->inode);      /* 从根节点向下刷写节点 */
```

设计关键：数据的完整性和一致性。

3. 创建文件/文件夹（newfs_mkdir, newfs_mknod）

分配必要的索引节点和数据块，更新位图和索引节点表。

在创建的过程中，首先要完成路径的解析，sfs_lookup 函数如下：

所有的路径解析都会从根目录开始，由超级块维护的根目录 dentry，读取根目录的 inode（在读取 inode 的时候会把数据块的内容读取进来，即所有子文件的 dentry_d），然后依次遍历所有子文件的 dentry，

对于寻找/hunt/bin.o:

找到文件名为 hunt 的 dentry，然后再读取 hunt 的 inode。然后根据 hunt 的 inode，再依次遍历所有子文件的 dentry，找到文件名为 bin.o 的 dentry。路径解析找到该文件，返回其 dentry。

```
while (fname)
{
    lvl++;
    if (dentry_cursor->inode == NULL) {           /* Cache机制 */
        newfs_read_inode(dentry_cursor, dentry_cursor->ino);
    }

    inode = dentry_cursor->inode;

    if (NEWFS_IS_REG(inode) && lvl < total_lvl) {
        NEWFS_DBG("[%s] not a dir\n", __func__);
        dentry_ret = inode->dentry;
        break;
    }
    if (NEWFS_IS_DIR(inode))
    {
        dentry_cursor = inode->dentrys;
        is_hit        = FALSE;

        while (dentry_cursor)
        {
            if (memcmp(dentry_cursor->fname, fname, strlen(fname)) == 0) {
                is_hit = TRUE;
                break;
            }
            dentry_cursor = dentry_cursor->brother;
        }

        if (!is_hit) {
            *is_find = FALSE;
            NEWFS_DBG("[%s] not found %s\n", __func__, fname);
            dentry_ret = inode->dentry;
            break;
        }

        if (is_hit && lvl == total_lvl) {
            *is_find = TRUE;
            dentry_ret = dentry_cursor;
            break;
        }
    }
    fname = strtok(NULL, "/");
}
```

接着创建新的 dentry 结构，添加到父目录：

当需要新增加一个 dentry 到父目录时，采用头插法。只需要修改父目录 inode 中的指针指向新增的 dentry 结构，新增的 dentry 的兄弟指针指向原来第一个子文件 dentry 即可。

```
int newfs_alloc_dentry(struct newfs_inode* inode, struct newfs_dentry* dentry) {
    if (inode->dentrys == NULL) {
        inode->dentrys = dentry;
    }
    else {
        dentry->brother = inode->dentrys;
        inode->dentrys = dentry;
    }
    inode->dir_cnt++;
    // 插入是size增加
    inode->size += sizeof(struct newfs_dentry);
    return inode->dir_cnt;
}
```

最后分配新的索引节点 inode：

```c
struct newfs_inode* newfs_alloc_inode(struct newfs_dentry * dentry) {
    struct newfs_inode* inode;
    int byte_cursor = 0;
    int bit_cursor  = 0;
    int ino_cursor  = 0;
    boolean is_find_free_entry = FALSE;

    for (byte_cursor = 0; byte_cursor < NEWFS_BLKS_SZ(super.ino_map_blks);
         byte_cursor++)
    {
        for (bit_cursor = 0; bit_cursor < UINT8_BITS; bit_cursor++) {
            if((super.map_inode[byte_cursor] & (0x1 << bit_cursor)) == 0) {
                                                    /* 当前ino_cursor位置空闲 */
                super.map_inode[byte_cursor] |= (0x1 << bit_cursor);
                is_find_free_entry = TRUE;
                break;
            }
            ino_cursor++;
        }
        if (is_find_free_entry) {
            break;
        }
    }

    if (!is_find_free_entry || ino_cursor == super.ino_max)
        return -NEWFS_ERROR_NOSPACE;

    inode = (struct newfs_inode*)malloc(sizeof(struct newfs_inode));
    inode->ino  = ino_cursor;
    inode->size = 0;
                                                    /* dentry指向inode */
    dentry->inode = inode;
    dentry->ino   = inode->ino;
                                                    /* inode指回dentry */
    inode->dentry = dentry;

    inode->dir_cnt = 0;
    inode->dentrys = NULL;

    for (int i = 0; i < NEWFS_DATA_PER_FILE; i++) {
        inode->data[i] = (uint8_t *)malloc(NEWFS_BLK_SZ());
        inode->blk_no[i] = -1;
    }

    return inode;
}
```

mkdir 与 mknod 函数如下所示：
设计关键是正确管理空间分配和释放。

```c
int newfs_mkdir(const char* path, mode_t mode) {
    /* TODO: 解析路径，创建目录 */
    (void)mode;
    boolean is_find, is_root;
    char* fname;
    struct newfs_dentry* last_dentry = newfs_lookup(path, &is_find, &is_root);
    struct newfs_dentry* dentry;
    struct newfs_inode*  inode;
    if (is_find) {
        return -NEWFS_ERROR_EXISTS;
    }

    if (NEWFS_IS_REG(last_dentry->inode)) {
        return -NEWFS_ERROR_UNSUPPORTED;
    }

    fname  = newfs_get_fname(path);
    dentry = new_dentry(fname, NEWFS_DIR);

    dentry->parent = last_dentry;


    inode  = newfs_alloc_inode(dentry);
    dentry->inode = inode;

    // 给新的索引结点分配第一个数据块
    if(newfs_alloc_datablk(dentry) < 0)
        printf("新增数据块失败");
    newfs_alloc_dentry(last_dentry->inode, dentry);
    // 我们可能还需要为新增的dentry预先申请一个新的父目录的数据块来供对应的
    // dentry d写回磁盘时使用（如果父目录原来申请的数据块已经放满了）
    int size_aligned_before = NEWFS_ROUND_UP((last_dentry->inode->size), NEWFS_BLK_SZ());
    int size_aligned_after = NEWFS_ROUND_UP((last_dentry->inode->size + sizeof(struct newfs_dentry)), NEWFS_BLK_SZ());
    if(size_aligned_after != size_aligned_before) {
        // 需要给父目录增加新的数据块
        if(newfs_alloc_datablk(last_dentry) < 0)
            printf("父目录新增数据块失败");
    }
    return 0;
}
```

```c
int newfs_mknod(const char* path, mode_t mode, dev_t dev) {
    /* TODO: 解析路径，并创建相应的文件 */
    boolean is_find, is_root;

    struct newfs_dentry* last_dentry = newfs_lookup(path, &is_find, &is_root);
    struct newfs_dentry* dentry;
    struct newfs_inode* inode;
    char* fname;

    if (is_find == TRUE) {
        return -NEWFS_ERROR_EXISTS;
    }

    fname = newfs_get_fname(path);

    if (S_ISREG(mode)) {
        dentry = new_dentry(fname, NEWFS_REG_FILE);
    }
    else if (S_ISDIR(mode)) {
        dentry = new_dentry(fname, NEWFS_DIR);
    }
    else {
        dentry = new_dentry(fname, NEWFS_REG_FILE);
    }
    dentry->parent = last_dentry;

    inode = newfs_alloc_inode(dentry);
    dentry->inode = inode;

    newfs_alloc_dentry(last_dentry->inode, dentry);

    // 我们可能还需要为新增的dentry预先申请一个新的父目录的数据块
    int size_aligned_before = NEWFS_ROUND_UP((last_dentry->inode->size), NEWFS_BLK_SZ());
    int size_aligned_after = NEWFS_ROUND_UP((last_dentry->inode->size + sizeof(struct newfs_dentry)), NEWFS_BLK_SZ());
    if(size_aligned_after != size_aligned_before) {
        // 需要给父目录增加新的数据块
        if(newfs_alloc_datablk(last_dentry) < 0)
            printf("父目录新增数据块失败");
    }
    return 0;
}
```

4. 查看文件夹下的文件（newfs_readdir）

遍历目录的索引节点，获取文件列表。

- 首先完成获取文件或目录的属性的函数：

```c
int newfs_getattr(const char* path, struct stat * newfs_stat) {
    /* TODO: 解析路径，获取Inode，填充newfs_stat，可参考/fs/simplefs/NEWFS.c的NEWFS_getattr()函数实现 */
    boolean is_find, is_root;

    /* 路径解析 */
    struct newfs_dentry* dentry = newfs_lookup(path, &is_find, &is_root);
    if (is_find == FALSE) {
        return -NEWFS_ERROR_NOTFOUND;
    }
    /* 结构体填充 */
    if (NEWFS_IS_DIR(dentry->inode)) {
        newfs_stat->st_mode = S_IFDIR | NEWFS_DEFAULT_PERM;
        newfs_stat->st_size = dentry->inode->dir_cnt * sizeof(struct newfs_dentry_d);
    }
    else if (NEWFS_IS_REG(dentry->inode)) {
        newfs_stat->st_mode = S_IFREG | NEWFS_DEFAULT_PERM;
        newfs_stat->st_size = dentry->inode->size;
    }
    else if (NEWFS_IS_SYM_LINK(dentry->inode)) {
        newfs_stat->st_mode = S_IFLNK | NEWFS_DEFAULT_PERM;
        newfs_stat->st_size = dentry->inode->size;
    }

    newfs_stat->st_nlink = 1;
    newfs_stat->st_uid   = getuid();
    newfs_stat->st_gid   = getgid();
    newfs_stat->st_atime   = time(NULL);
    newfs_stat->st_mtime   = time(NULL);
    newfs_stat->st_blksize = NEWFS_BLK_SZ();

    if (is_root) {
        newfs_stat->st_size = super.sz_usage;
        newfs_stat->st_blocks = NEWFS_DISK_SZ() / NEWFS_BLK_SZ();
        newfs_stat->st_nlink  = 2;      /* !特殊，根目录link数为2 */
    }
    return 0;
}
```

然后是读取目录：

读取目录的函数主要是将子文件的文件名填充到指定的缓冲区即可。

读取目录函数主要负责的是：解析父目录路径，并根据子文件偏 offset 获取到对应的子文件文件名 。

```c
int newfs_readdir(const char * path, void * buf, fuse_fill_dir_t filler, off_t offset,
                  struct fuse_file_info * fi) {
    /* TODO: 解析路径，获取目录的Inode，并读取目录项，利用filler填充到buf，可参考/fs/simplefs/NEWFS.c的NEWFS_readdir()函数实现 */
    boolean is_find, is_root;
    int     cur_dir = offset;

    struct newfs_dentry* dentry = newfs_lookup(path, &is_find, &is_root);
    struct newfs_dentry* sub_dentry;
    struct newfs_inode* inode;
    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;
    }
    return -NEWFS_ERROR_NONE;
}
```

设计关键：高效地遍历和读取目录结构。

3、 *实验特色*

实验中你认为自己实现的比较有特色的部分

特色一：高效的空间管理

使用位图来管理索引节点和数据块的分配，降低了空间的碎片化，提高了空间利用率。

特色二：

在关键操作中增加了错误处理逻辑，确保文件系统的稳定性和数据的一致性。

二、用户手册

实现的文件系统中的所有命令使用方式

卸载文件系统

- 命令: fusermount -u <挂载点>
- 说明: 此命令将会卸载您的文件系统。<挂载点>是文件系统当前挂载的目录。

创建文件夹

- 命令: mkdir <路径>
- 说明: 在指定路径创建一个新的目录。<路径>是您希望创建目录的路径。

创建文件

- 命令: touch <路径>
- 说明: 在指定路径创建一个新的空文件。<路径>是您希望创建文件的路径。

查看文件夹下的文件

- 命令: ls <目录>
- 说明: 显示指定目录下的文件和文件夹。<目录>是您希望查看的目录路径。

注意事项

- 确保您在执行命令之前已经正确地挂载了文件系统。
- 在执行任何写操作之前，请确保您有足够的权限。
- 使用 fusermount -u 来安全地卸载文件系统，避免数据丢失。

三、实验收获和建议

实验中的收获、感受、问题、建议等。

- 实验收获:

深入理解文件系统结构：通过实现一个青春版的 EXT2 文件系统，更深入地理解文件系统的内部结构和工作原理，包括超级块、索引节点、数据块等关键部分的功能和相互关系。

实践操作系统理论：此实验帮助我将操作系统课程中的理论知识转化为实践经验，特别是在文件系统、磁盘管理等方面。

提高编程能力：在编写和调试底层代码的过程中，您的 C 语言编程能力、调试技巧和问题解决能力得到了锻炼。

了解 Linux 内核和 FUSE：通过使用 FUSE (Filesystem in Userspace) 框架，您对 Linux 内核中的文件系统部分及用户空间和内核空间的交互有了更深刻的认识。

IO 操作实践：实验中涉及到的磁盘 IO 操作使您对文件系统如何与硬件层交互有了实际的体验。

- 实验感受：

挑战性：实现文件系统是一项复杂且富有挑战性的任务，特别是当涉及到底层的数据结构和硬件交互时。

满足感：成功运行和测试文件系统带来了成就感。

学习曲线陡峭：在实验的初期，由于需要了解许多新的概念和技术，感到有些困难。

- 遇到的问题

调试困难：在底层编程时，调试可能会比较困难，特别是当遇到内存泄漏或数据不一致时。

理论与实践差异：理论知识与实际操作之间可能存在差异，实际操作中可能会遇到一些未在理论课上讲解的问题。

四、参考资料

实验过程中查找的信息和资料

实验指导书 http://hitsz-cslab.gitee.io/os-labs/lab5/part1/