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：计科 2 班

学生学号：1190303311

学生姓名：王志军

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

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

Newfs 文件系统的设计参照 simplefs 文件系统的结构，通过阅读 simplefs 代码发现其只有索引位图而没有数据位图，原因在于其架构是将文件聚簇存储的，也就是在磁盘中，文件的索引块和数据块相邻存放，并且每个文件的数据块数是固定的，这样就没必要使用数据位图；

为了添加数据位图并且使其有作用，考虑文件的索引块在磁盘中聚簇排序存储，而文件的数据块采用乱序存储的方式，以此来加入数据位图的优势，不必为文件分配连续的数据块，方便起见，newfs 文件系统给每个文件分配的数据块位数大小也是固定的。

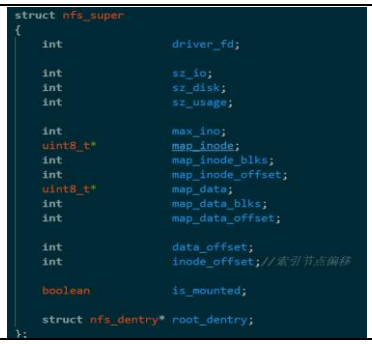
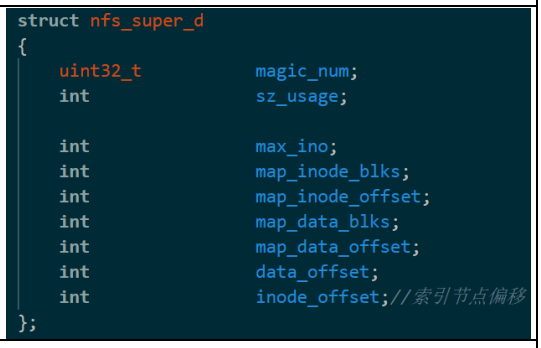
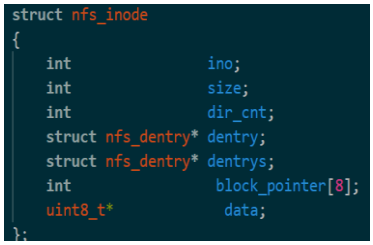
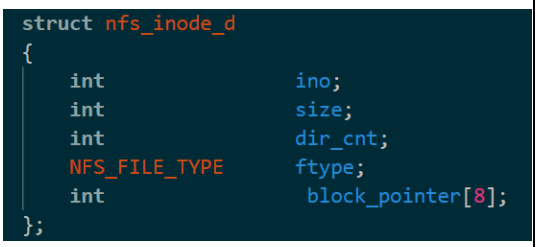
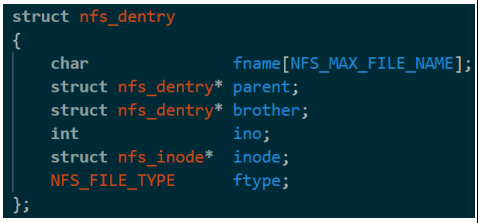
2、 功能详细说明

每个功能点的详细说明（关键的数据结构、代码、流程等）

2．1、数据结构

Newfs 文件系统的数据结构包括磁盘数据结构和内存数据结构两部分，分别用于磁盘文件的存储读取以及文件在内存中驻留；

根据实验指导书给出的文件系统结构，设计了超级块、索引块和条目三种数据结构：

	内存数据结构	磁盘数据结构
超级块		
索引		
条目		

根据分析，文件系统通过索引和条目结构来实现树形索引，即每个文件对应唯一一个索引节点，一个索引节点指向一个文件的数据块，数据块中可以是文件数据，也可以是条目项，条目项指向一个索引节点，也就是子目录的索引节点。

除此之外是定义的一些宏，用来计算地址或者对齐等功能。

2.2、功能

2.2.1 挂载

挂载函数由 nfs_mount()实现，作用有读取超级块，并将磁盘信息传递给内存中的全局变量超级块，读出位图；如果是第一次挂载需要计算资源信息；最终读出根目录存储在内存中；

计算磁盘属性，计算最大节点个数和索引、数据位图所占的位数，得出位图所占的块数，并计算位图和数据区的偏移：

```
super_blks = NFS_ROUND_UP(sizeof(struct nfs_super_d), NFS_IO_SZ()) / NFS_IO_SZ();
//总共可以容纳多少个inode
inode_num  =  NFS_DISK_SZ() / ((NFS_DATA_PER_FILE + NFS_INODE_PER_FILE) * NFS_IO_SZ());
//inode位图占多少块
data_num = inode_num * NFS_DATA_PER_FILE;
map_inode_blks = NFS_ROUND_UP(NFS_ROUND_UP(inode_num, UINT32_BITS), NFS_IO_SZ())
                 / NFS_IO_SZ();
map_data_blks = NFS_ROUND_UP(NFS_ROUND_UP(data_num, UINT32_BITS), NFS_IO_SZ())
                / NFS_IO_SZ();
                                                    /* 布局layout */
nfs_super.max_ino = (inode_num - super_blks - map_inode_blks - map_data_blks);
nfs_super_d.map_inode_offset = NFS_SUPER_OFS + NFS_BLKS_SZ(super_blks);
nfs_super_d.map_data_offset = nfs_super_d.map_inode_offset + NFS_BLKS_SZ(map_inode_blks);
nfs_super_d.inode_offset = nfs_super_d.map_data_offset + NFS_BLKS_SZ(map_data_blks);//索引
nfs_super_d.data_offset = nfs_super_d.inode_offset + nfs_super.max_ino*NFS_BLKS_SZ(1);//数
nfs_super_d.map_inode_blks  = map_inode_blks;
nfs_super_d.map_data_blks = map_data_blks;
nfs_super_d.sz_usage    = 0;
```

将磁盘信息储存到内存数据结构当中：

```
nfs_super.map_inode = (uint8_t *)malloc(NFS_BLKS_SZ(nfs_super_d.map_inode_blks));
nfs_super.map_data = (uint8_t *)malloc(NFS_BLKS_SZ(nfs_super_d.map_data_blks));
nfs_super.map_inode_blks = nfs_super_d.map_inode_blks;
nfs_super.map_inode_offset = nfs_super_d.map_inode_offset;
nfs_super.data_offset = nfs_super_d.data_offset;
nfs_super.inode_offset = nfs_super_d.inode_offset;
nfs_super.map_data_blks = nfs_super_d.map_data_blks;
nfs_super.map_data_offset = nfs_super_d.map_data_offset;

if (nfs_driver_read(nfs_super_d.map_inode_offset, (uint8_t *)(nfs_super.map_inode),
                    NFS_BLKS_SZ(nfs_super_d.map_inode_blks)) != NFS_ERROR_NONE) {
    return -NFS_ERROR_IO;
}

if (nfs_driver_read(nfs_super_d.map_data_offset, (uint8_t *)(nfs_super.map_data),
                    NFS_BLKS_SZ(nfs_super_d.map_data_blks)) != NFS_ERROR_NONE) {
    return -NFS_ERROR_IO;
}
```

最后 read 根节点，也就是在内存中存储根节点下的目录项，存储在内存中。

2.2.2 卸载

卸载需要将超级块和位图写回磁盘，同时需要从根节点向下递归将所有文件和目录写回磁盘：

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

nfs_super_d.magic_num           = NFS_MAGIC_NUM;
nfs_super_d.map_inode_blks      = nfs_super.map_inode_blks;
nfs_super_d.map_inode_offset    = nfs_super.map_inode_offset;
nfs_super_d.data_offset         = nfs_super.data_offset;
nfs_super_d.inode_offset         = nfs_super.inode_offset;
nfs_super_d.map_data_blks      = nfs_super.map_data_blks;
nfs_super_d.map_data_offset    = nfs_super.map_data_offset;
nfs_super_d.sz_usage            = nfs_super.sz_usage;

if (nfs_driver_write(NFS_SUPER_OFS, (uint8_t *)&nfs_super_d,
                     sizeof(struct nfs_super_d)) != NFS_ERROR_NONE) {
    return -NFS_ERROR_IO;
}

if (nfs_driver_write(nfs_super_d.map_inode_offset, (uint8_t *)(nfs_super.map_inode
                     NFS_BLKS_SZ(nfs_super_d.map_inode_blks)) != NFS_ERROR_NONE) {
    return -NFS_ERROR_IO;
}

if (nfs_driver_write(nfs_super_d.map_data_offset, (uint8_t *)(nfs_super.map_data),
                     NFS_BLKS_SZ(nfs_super_d.map_data_blks)) != NFS_ERROR_NONE) {
    return -NFS_ERROR_IO;
```

2.2.3 获取文件属性

这一步和指导书给的例子相似，调用 lookup 函数获取路径最低一级文件的 dentry，然后将其属性返回，重点在 lookup 函数；

Lookup 函数先对路径进行解析，并从根节点开始遍历，在每一级的目录项当中寻找与路径这一级名对应的项，存在则继续向下直到找到路径所指文件，如果找到了这个文件的 dentry，还需要调用 nfs_read_inode()函数从磁盘读取它的下一级文件，存储在内存中；

```
if (dentry_ret->inode == NULL) {
    dentry_ret->inode = nfs_read_inode(dentry_ret, dentry_ret->ino);
}
```

2.2.4 建立文件

函数名 mknod，先调用 lookup 看文件是否已经存在，不存在则进行创建：

```
struct nfs_dentry* last_dentry = nfs_lookup(path, &is_find, &is_root);
struct nfs_dentry* dentry;
struct nfs_inode* inode;
char* fname;

if (is_find == TRUE) {//该文件已经存在
    return -NFS_ERROR_EXISTS;
}

fname = nfs_get_fname(path);

if (S_ISREG(mode)) {
    dentry = new_dentry(fname, REG_FILE);
}
else if (S_ISDIR(mode)) {
    dentry = new_dentry(fname, DIR);
}
//上级目录的dentry指向上级目录的inode，inode指向这一级的dentry，再指向这一
dentry->parent = last_dentry;
inode = nfs_alloc_inode(dentry);
nfs_alloc_dentry(last_dentry->inode, dentry);
```

Lookup 当路径不存在时返回的是路径最后存在一级的 dentry，也就是要创建文件的父目录，需要将其和新创建文件的 dentry 和 inode 联系起来；

2.2.5 建立目录

和创建文件相似，除了需要创建的 dentry 类型是 dir 之外还需要判断父母录是否是文件，是则无法在文件下创建目录，报错：

```
//最后一级是文件，无法在文件下创建目录
if (NFS_IS_REG(last_dentry->inode)) {
    return -NFS_ERROR_UNSUPPORTED;
}
```

2.2.6 读取目录项

这一步是输出路径下的所有目录项，ls 的时候有循环调用，先 lookup 解析路径，如果存在就通过 fuse 的 filler 函数将目录项的名字写入到 buf 中返回；

```
int nfs_readdir(const char * path, void * buf, fuse_fill_dir_t filler,
                struct fuse_file_info * fi) {
    boolean is_find, is_root;
    int     cur_dir = offset;

    struct nfs_dentry* dentry = nfs_lookup(path, &is_find, &is_root);
    struct nfs_dentry* sub_dentry;
    struct nfs_inode* inode;
    if (is_find) {
        inode = dentry->inode;
        sub_dentry = nfs_get_dentry(inode, cur_dir);
        if (sub_dentry) {
            filler(buf, sub_dentry->fname, NULL, ++offset);
        }
        return NFS_ERROR_NONE;
    }
    return -NFS_ERROR_NOTFOUND;
}
```

其中 nfs_get_dentry 函数是返回 inode 的第 cur_dir 个 dentry：

```
    struct nfs_dentry* dentry_cursor = inode->dentrys;
    int    cnt = 0;
    while (dentry_cursor)
    {
        if (dir == cnt) {
            return dentry_cursor;
        }
        cnt++;
        dentry_cursor = dentry_cursor->brother;
    }
    return NULL;
```

通过 fuse，最终可以实现创建文件和目录，ls 查看文件目录，挂载和卸载等操作。

3、 实验特色

实验中你认为自己实现的比较有特色的部分

3.1、数据位图的实现

在为文件分配索引位图时，进行数据块的分配，同时修改数据位图，数据块分配时，遍历数据位图，找到足够的空闲块之后修改位图，并将这些块的块号存储，inode 的内存数据结构当中既有 data 指针，又有 blockpointer 存储在磁盘上分配的数据块的块号，可以实现乱序分配数据块：

```
for (data_byte_cursor = 0; data_byte_cursor < NFS_BLKS_SZ(nfs_super.map_data_blks); data_byte_cursor++)
{
    for(data_bit_cursor = 0; data_bit_cursor<UINT8_BITS; data_bit_cursor++)
    {
        if((nfs_super.map_data[data_byte_cursor] & (0x1 << inode_bit_cursor)) == 0)
        {
            nfs_super.map_inode[data_byte_cursor] |= (0x1 << data_bit_cursor);
            inode->block_pointer[blk_num] = data_cursor;
            blk_num++;
            if(blk_num == NFS_DATA_PER_FILE)
            {
                is_find_free_data = TRUE;
                break;
            }
        }
        data_cursor++;
    }
    if(is_find_free_data)
        break;
}
```

3.2、数据读取写回

读取目录时，需要根据 dir_cnt 目录数从数据块中读出所有的 dentry：

```
if (NFS_IS_DIR(inode)) {
    dir_cnt = inode_d.dir_cnt;
    for(int k = 0; k < NFS_DATA_PER_FILE; k++){
        int offset = NFS_DATA_OFS(inode_d.block_pointer[k]);
        int blk_offset = 0;
        if(i == dir_cnt || dir_cnt == 0)
            break;
        while(blk_offset < NFS_BLKS_SZ(1)){
            if(i == dir_cnt || dir_cnt == 0)
                break;
            if (nfs_driver_read(offset + blk_offset, (uint8_t *)&dentry_d,
                                sizeof(struct nfs_dentry_d)) != NFS_ERROR_NONE) {
                NFS_DBG("[%s] io error\n", __func__);
                return NULL;
            }
            sub_dentry = new_dentry(dentry_d.fname, dentry_d.ftype);
            sub_dentry->parent = inode->dentry;
            sub_dentry->ino    = dentry_d.ino;
            nfs_alloc_dentry(inode, sub_dentry);
            blk_offset+=sizeof(struct nfs_dentry_d);
            i++;
        }
```

写回时限制写回大小，不能越界。

```
else if (NFS_IS_REG(inode)) {
    int data_oft = 0, k=0;
    while(((inode->size-data_oft) >= NFS_BLKS_SZ(1)) && k<8){
        if (nfs_driver_write(NFS_DATA_OFS(inode->block_pointer[k]), inode->data+data_oft,
                             NFS_BLKS_SZ(1)) != NFS_ERROR_NONE) {
            NFS_DBG("[%s] io error\n", __func__);
            return -NFS_ERROR_IO;
        }
        k++;
        data_oft+=NFS_BLKS_SZ(1);
    }

    if(k<8 && ((inode->size-data_oft) < NFS_BLKS_SZ(1))){
        if (nfs_driver_write(NFS_DATA_OFS(inode->block_pointer[k]),
        inode->data+data_oft, inode->size-data_oft) != NFS_ERROR_NONE) {
            NFS_DBG("[%s] io error\n", __func__);
            return -NFS_ERROR_IO;
        }
    }
}
```

二、用户手册

实现的文件系统中的所有命令使用方式

查看当前目录下文件：ls

创建文件：touch filename

创建目录：mkdir name

进入目录：cd dirname

挂载：./build/newfs --device=/home/guests/1190303311/ddriver -f -d -s ./tests/mnt/

卸载：fusermount -u ./tests/mnt

三、实验收获和建议

操作系统整个实验课程过程中的收获、感受、问题、建议等。

做完所有实验感觉收获了很多，课堂上讲的理论还是要做实验才能完全理解。

操作系统实验是目前遇到过的最难的实验课了，前几次的实验难度还好，最后两次的难度较大，而且考察的更加全面，之前在本部上过 CSAPP（计算机系统），有些类似的实验。本次文件系统设计的实验给了很多时间，刚开始我是很懵，完全不知道从哪里开始下手，设计类的实验是我认为最难的，后来看懂了 simplefs 的实现，修改了一下存储结构，也算是完成了；

实验很有挑战性，实验课设置也很好。

四、参考资料

实验过程中查找的信息和资料

操作系统课程课件