ICS II FS Lab 报告

中国人民大学 sheriyuo

1 组织结构

1.1 总体架构

仿照 VSFS 的设计，采用如下结构：

SuperBlock	Inode Bitmap	Data Bitmap	Inode Block	Data Block

系统共包含 65536 个块，每个块的大小 4KB，共 256MB 大小。

第 1 个块为 SuperBlock，第 2 个块为 Inode Bitmap，第 3, 4 个块为 Data Bitmap。

文件系统需要支持至少 32768 个文件及目录，对应 1 个 Bitmap 可以存储 $4096 \times 8 = 32768$ 个 Inode 信息。

对于一个 Inode，采用 12 个直接指针和 2 个间接指针，间接指针首部用于存储指针数，可支持最大 $(12 + 2 \times 1023) \times 4\text{KB} = 8.04\text{MB}$ 大小的文件。

```
typedef struct INode {
    mode_t mode;
    size_t size;
    time_t atime, mtime, ctime;
    int blockNum, block[DIRECT_POINTER], indirect[2];
} INode;
```

一个 Inode 大小为 96 字节，一个 Block 可以存储 42 个 Inode，共需要 $32768 \div 42 = 780$ 个 Inode Block。

对于 Data Block，有效块数为 $65536 - 4 - 780 = 64752$，对应 2 个 Data Bitmap。

1.2 文件架构

对于普通文件与目录文件，均采用 `File` 结构记录其对应的 `Inode` 编号和文件名，存储在父目录 `Directory` 块中，使用 `Inode` 的 `mode` 参数来区分类型。

```
typedef struct File {
    int node;
    char name[FILE_NAME];
} File;

typedef struct Directory {
    int size;
    File file[FILE_PER_BLOCK];
} Directory;
```

一个 `File` 结构大小为 28 字节，一个 Block 可以存储 $4092 \div 28 = 146$ 个 `File`，对应一个 `Inode` 可以存储 $(12 + 2 \times 1023) \times 146 = 300468$ 个 `File`。

1.3 SuperBlock

采用以下结构来存储 SuperBlock，在调用 `mkfs` 函数时初始化并存储在磁盘中。

```
typedef struct SuperBlock {
    unsigned long blockSize, fileName;
    fsblkcnt_t blockNum, fileNode, freeBlock, freeNode;
} SuperBlock;
```

调用 `fs_statfs` 函数时从磁盘读取 SuperBlock 来获取文件系统信息，并在分配/回收 Inode/Block 时更新。

2 具体实现

2.1 辅助函数

2.1.1 文件寻址

采用以下函数实现文件寻址，由 `path` 得到其文件名与对应的 `Inode`，核心实现为使用 `strtok` 函数来处理字符串。

```
INode getINodeByNum(int num);
int getNumByName(int fNum, const char *name);
int getNumByPath(const char *path);
void getNameByPath(const char *path, char *name);
int getFatherNumByPath(const char *path);
```

2.1.2 磁盘读写

使用 `disk_read` 函数读取 `char *` 的字节信息，并转换为对应的结构体指针。使用 `disk_write` 函数将信息写回磁盘中。

```
SuperBlock readSuperBlock();
void writeSuperBlock(SuperBlock blk);
void modifySuperBlock(int freeNode, int freeBlock);
void readDirectory(int num, Directory *dir);
int initIndirect(INode *node, int i);
void readIndirect(INode node, int *data, int i);
void writeINode(int num, INode inode);
```

2.1.3 文件新建/删除

通过目录寻址得到的文件 `Inode` 信息判断是修改 `Block` 还是 `Indirect` 指针，在对应的 `Block` 上进行修改，并更新对应的 Bitmap。

对于文件的操作均先在父目录的 `Directory` 结构中进行，更新 `Inode` 信息，判断修改作用块，需要时分配 `Indirect` 指针并保存修改。

```
int findFree(int start);
int findFreeBlock();
void initINode(INode *node, mode_t mode);
void removeData(int num);
void removeINode(INode node, int num);
int addFile2Block(File file, int num);
int addFile2Indirect(File file, INode fa, int i);
int addFile2Dir(File file, int num);
int removeFileInDir(Directory *dir, const char *name);
int removeFile(int num, const char *name);
```

2.1.4 文件 I/O

`fs_read` 函数与 `fs_write` 函数的调用有固定的 `buffer` 大小，根据其情况（直接/间接指针）采用分块式的对 `Block` 的 I/O。

```
int truncateIndirect(INode *node, size_t size);
int readBlock(INode node, char *buffer, int blk);
int readStartBlock(INode node, char *buffer, int blk, int offset);
int readEndBlock(INode node, char *buffer, int blk, int size);
int writeBlock(INode node, const char *buffer, int blk, int offset, int size);
```

2.2 mkfs

初始化 SuperBlock 后写入磁盘中，新建并初始化 Bitmap 块，初始化 mnt 对应的 Inode 0，同时作为目录寻址的根节点。

2.3 fs_statfs

从磁盘读取 SuperBlock，返回对应信息。

2.4 fs_getattr

寻址求出对应 Inode，返回对应信息。

2.5 fs_readdir

寻址求出对应 Inode，在所有指针对应块的 Directory 结构中查找并输出信息。

2.6 fs_mknod, fs_mkdir

在 Bitmap 中寻找空闲 Inode，分配编号及寻址得到的文件名，并在父目录中添加该文件。

普通文件与目录文件的区别在于初始化 Inode 时的 mode 属性（REGMODE/DIRMODE）。

2.7 fs_unlink, fs_rmdir

寻址求出对应 Inode，在磁盘中删除 Inode 信息，并在父目录下删除此 File。

删除文件时若清空了一个指针所对应的 Directory Block，则删除此 Block，将原指针赋值为 -1。在添加文件时优先顺序遍历所有指针，为空指针重新分配块。

2.8 fs_rename

寻址求出对应 Inode，在旧父目录下删除此 File，并通过新目录修改其文件名，在新父目录中添加该文件。

2.9 fs_truncate

判断所需要的块大小，若只需要直接指针，则直接修改；否则，修改间接指针对应块，分类讨论修改的情况，对于间接指针的修改均保存在磁盘中。

2.10 fs_read

寻址求出对应 Inode，计算 [offset, offset+size] 对应的分块信息 $[L, R]$，对于首块、中间块（完整块）、尾块按顺序进行读入，每次处理完一个块更新当前 buffer 的 offset。

2.11 fs_write

读文件的 buffer 最大为 128KB，而写文件的 buffer 最大为 4KB，即在 cp 操作时固定以一个 Block 为单位进行写入。

因此只需要实现查找当前 offset 所在块，对于单个块进行写入。

3 实验结果

可以通过所有测试数据，其中本地测试未通过的 2 组数据为实验环境所导致（时区、用户信息）。

```
sheriyuo@ROG-Strix-6P:~/fslab-handout$ ./run.sh
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 0 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 1 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 2 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 3 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 4 failed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 5 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 6 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 7 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 8 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 9 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 10 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 11 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 12 failed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 13 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 14 passed
fusermount: entry for /home/sheriyuo/fslab-handout/mnt not found in /etc/mtab
traces 15 passed
```

批量化测试的逻辑如下，比较忽略换行符：

```
compare_files() {
    local file1=$1
    local file2=$2
    diff <(tr -d '\n' < "$file1") <(tr -d '\n' < "$file2") > /dev/null
}

for i in {0..15}
do
    make -s mount
```

```
    ./traces/"$i".sh > log/"$i".ans
    make -s umount

    if compare_files log/"$i".ans traces/"$i".ans; then
        echo "traces $i passed"
    else
        echo "traces $i failed"
fi
```