实验报告

文件系统组织结构

总体架构

我参考**VSFS**文件系统的设计，设计了如下的组织形式：

```
-------------------------------------------------------------------------------
-----
| SuperBlock | Inode Bitmap| Block Bitmap1 | Block Bitmap2 | Inode Block 1| ... |
-------------------------------------------------------------------------------
-----
```

```
-------------------------------------------------
Inode Block n | Data Block 1 | ... | Data Block m|
-------------------------------------------------
```

该系统一共包含65536个块，每个块的大小时4KB，共256MB的大小。

SuperBlock

我将第一个块设置为SuperBlock，存储文件系统的基本信息，该SuperBlock组织结构如下：

```
typedef struct SuperBlock {
    unsigned long blockSize; // block size的大小
    fsblkcnt_t blockNum; // block size 的数量
    fsblkcnt_t blockFree; // 空闲的block数量
    fsblkcnt_t fileNode; // 文件节点额数量
    fsblkcnt_t freeNode; // 空闲节点的数量
    unsigned long fileName; // 文件名长度上限
}SuperBlock;
```

在文件系统中中调用fs_statfs函数，可以查询文件系统的基本信息。

Inode

在第二个块中我存储管理Inode的数据结构，这里我选取Bitmap对Inode进行管理。在该文件系统中，需要吃吃32768个文件及目录，所以至少需要32768个Inode管理文件。因此，我将Bitmap的大小设置为$1024 * 4 = 4096$字节，即一个Block，此时Bitmap可以存储$4096 * 8 = 32768$个bit位，每一位代表这个Inode节点是否被占用。Inode组织结构如下：

```
typedef struct INode {
    mode_t mode;
    off_t size;
    time_t accessTime;
    time_t modifyTime;
    time_t changeTime;
    int blockNum;
    int blocks[DIRECT_POINTER];
    int indirectPointer[2];
}INode;
```

在每一个Inode块中，我存储12个直接指针，指向Data Block数据块，并存储2个简洁指针，指向两个数据块，每个间接数据块的组织形式如下：

```
-----------------------------------------------------
| size | pointer 1 | pointer 2 | ... | pointer 1023 |
-----------------------------------------------------
```

我将其组织为一个int数组形式，其中首位为该间接指针块存储了几个间接指针，之后存储指针指向数据块。因此，每个Inode实际可以存储$12 + 1023 * 2 = 2058$个指向数据块的指针，因此单个文件最大为$2058 * 4KB > 8MB$。每个Inode块共大小为96字节，每一个块可以存储$4096/96 = 42$个Inode，一共需要$32768/42 = 780$个块用于存储Inode。

Data Block

在第三和第四个块中我存储管理Data块的Bitmap，管理存储文件的数据块。实际有效的数据块数量为$65536 - 4 - 780 = 64752$，所以该文件系统的实际使用空间大小为$64752 * 4KB = 252.93MB$，满足要求。同时共需要两个块作为Data Bitmap块管理数据块。

Directory

对于目录文件，我将其其他行为设置的与普通文件相同，同时我将存储文件内容的数据块管理为目录的形式，定义如下：

```
typedef struct Pair {
    int node;
    char fileName[MAX_FILE_NAME];
}Pair;

typedef struct Directory {
    int size;
    Pair pairList[DIC_PER_BLOCK];
}Directory;
```

我在首部存储该目录下一共存在多少个文件，之后我将文件信息存储为pair形式，包含其文件名与Inode编号。之后在插入文件与删除文件时对Directory精进行管理。

辅助函数

与读写块相关

```
// 读SuperBlock的值
SuperBlock readSuperBlock()

// 写SuperBlock的值
void writeSuperBlock(SuperBlock superBlock)

// 将第num个block以目录形式读出
void readDirectory(int num, Directory* dir)

// 初始化indirect指针
int initIndirect(INode* iNode, int t)

// 从iNode中读取出第t个indirect指针
void readIndirect(INode iNode, int* data, int t)
```

与处理Inode相关

```
// 找到空闲的iNode
int findFree(int start)

// 找到一个空闲块
int findFreeBlock()

// 通过编号获取INode
INode getINodeByNum(int num)

// 通过文件名获取iNode
int getINodeByName(int fatherNum, const char* name)

// 写iNode
```

```
void writeINode(int node, INode iNode)
```

与文件路径处理相关

```
// 通过路径获取iNode的编号
int getNumByPath(const char* origin)

// 通过路径获取文件名
void getNameByPath(const char* origin, char* name)

// 通过路径得到父节点的路径
int getFatherNumByPath(const char *origin)
```

与插入文件相关

```
// 将文件插入块中
int addFile2Block(Pair pair, int dataNum)

// 将文件插入indirect指针中
int addFile2Indirect(Pair pair, INode iNode, int t)

// 将文件信息加入目录文件中
int addFile2Dir(Pair pair, int num)
```

与删除文件相关

```
// 从bitmap中删除data
void removeData(int num)

// 从bitmap中删除iNode
void removeINode(INode iNode, int num)

// 在目录中删除文件
int removeFileinDir(Directory* dir, const char* name)

// 在父节点中删除文件
int removeFile(const char* name, int father)
```

与文件读写相关

```
// 在indirect指针中改变大小
int truncateIndirect(INode* iNode, off_t size)

// 读文件开头块
int readStartBlock(INode iNode, char* buffer, int startNode, int startOffset)

// 读文件结尾块
int readEndBlock(INode iNode, char* buffer, int endNode, int offset, int
endOffset)

// 写文件
int writeBlock(INode iNode, const char* buffer, int node, off_t off, int size)
```

实现函数细节

创建文件

首先我先找在Inode Bitmap中到一个空的Inode节点分配给待创建的文件，之后对Inode进行初始化并将初始化后的Inode节点写入Inode对应的数据块。再之后需要将分配的Inode编号和文件名组成一个pair插入父母目录中。在

删除文件

首选需要找到待删除文件的Inode节点，之后遍历Inode节点中指向的Data块并将资源释放。之后需要找到其父节点，并在其目录中存储的pair。

修改文件路径

首先需要在原路径中找到待修改文件的父节点，并在父节点中将需要修改的文件删除。之后将原Inode插入到新的路径中，此时不需要改变Inode中的文件信息，只需要改变存储其文件信息的pair的父节点。

改变文件大小

首先需要计算一共需要多少Data块存储此文件，即需要对文件Inode中存储的指针进行扩展。若需要点的Data块的大小小于12，则只需改变Inode的BlockNum值并对blocks中的直接指针进行分配；否则需要读出其Indirect块并增加其中的间接指针。

读常规文件

在读文件中，需要计算需要读的文件的开头块和结尾快与开头块与结尾块的offset，并对开头块和结尾块结合offset进行单独处理。之后遍历开头块与结尾块间的块，读取其Data Block到缓冲区中。

写常规文件

在写常规文件中，最初我将其于读文件的操作对照处理，之后我发现每一次写文件都是只会向文件中写入一个块，同时最大写入量为4KB。于是我简化文件写操作，计算其开始块并写入。

遇到的问题与解决方案

在本次实验，我在读文件与写文件中存在错误。我将写入二进制文件转化为十六进制文件后发现前端有一段数据一直为0并且其他写入的数据与原始数量居相同。通过写入0的部分，我初步判断在读取文件与写入文件中读取Inode Indirect前端部分存在bug。之后我把写入的内容与读出的内容打印在终端中检查，并没有检测中在哪一段中传入的值为0。之后我回看我的代码，发现我在blocks中存储的指针与Indirect中存储的指针下标不同，因为Indirect中首位需要存储Indirect指针的数量，所以存储指针的下标从1开始，因此遍历完blocks中的直接指针后需要将存储指针的编号减去11而不是12。我将该问题修复后读写问题得到解决。