fslab实验报告

文件系统设计

由于要求支持32768个文件及目录，刚好是一个块（4K）的位个数，因此我们可以用一个block来表示inode bitmap，同时我们使用2个block作为datablock bitmap，在使用到的时候我们可以使用一个数组来存储所需内容：

```
unsigned long long bitmap[512]; //4096 char and 32768 bits
```

对于每个inode，由于只需支持单个文件最大$8MB$，我们可以在每个inode中只使用一个indirect pointer，这个“指针”类型为unsigned short，这使得我们可以在一个inode内使用$4K \times 65535 = 256MB$的大小，满足文件要求的8MB大小要求，这样我们定义如下简化后的inode：

```
struct Inode
{
    unsigned int size;
    unsigned short indirpointer;
    bool directory;
    time_t atime;
    time_t ctime;
    time_t mtime;
};
```

由于对齐，每个Inode共占用32bytes，因而32768个inode需要$32768 \times 32bytes \div 4Kbytes = 256$，因此我们将第3-258号的块分给inode，从第259-65535号块开始表示真正的数据，这样也可以满足250MB的空间需求。综上，我们block的分配设计如下，用宏定义表示：

```
#define InodeBitMap 0
#define DataBitMap1 1
#define DataBitMap2 2
#define InodeStart 3
#define DataStart 259
#define InodeMax 32768
#define DataMax 65535
```

最后，对于目录文件，我们使用简单的文件名数组和文件inode来表示其内容，共占32bytes：

```
struct Dir
{
    char filename[30];
    unsigned short number;
};
```

函数设计

由于篇幅限制在4页以内，故函数只介绍功能，具体细节见代码。

辅助函数

- NewBlock
  分配一个新的数据块
- AllocateBlocks
  分配多个新的数据块
- DeleteBlock
  删除一个数据块
- DeleteBlocks
  删除多个数据块
- FindBlock
  查找是否有足够的数据块
- GetInode
  读一个inode
- WriteInode
  写一个inode
- NewInode
  分配一个新的inode
- DeleteInode
  删除一个inode
- FindInode
  查找是否有足够的inode
- InitInode
  初始化inode
- EraseInode
  销毁inode
- GetDir
  读一个目录
- WriteDir
  写一个目录
- MoreDir
  拓展一个目录
- FindDir
  查找一个目录
- Ptoi
  路径转文件inode

- Ptopi
  路径转文件的目录inode

主函数

- mkfs
  初始化根目录的 inode 即可。

- fs_getattr
  用Ptoi通过路径找到对应的inode，再读取信息。

- fs_readdir
  用Ptopi函数找到对应的inode，再用GetDir直接读取目录即可，之后再更新目录文件的atime。

- fs_read
  用Ptoi函数找到目标文件的inode，然后根据偏移量和读取长度进行读取，最后记得更新inode的atime。

- fs_mknod
  首先空间是否够用，若不够用则直接报错返回。随后使用Ptopi函数解析路径得到父目录的inode， 更新父目录的目录项，此处使用MoreDir函数检查新增一个目录项时父目录是否需要新分配一个数据块，随后为这个新分配的目录分配一个inode并调用InitInode函数，最后更新父目录的 atime，mtime，ctime。

- fs_mkdir
  同fs_mknod一致，不过创建的是目录文件。

- fs_rmdir
  类似fs_mknod，先用Ptopi解析路径得到父目录的inode，删除要求文件的目录项即可，之后更新父目录的 ctime 和mtime。

- fs_unlink
  与fs_rmdir一致，不再赘述。

- fs_rename
  通过Ptopi函数找到文件原位置和目标位置的父目录inode，查找该文件的目录项。若二者的目录项相同，则我们只需要在原位置的目录项中改名即可，否则我们需要在原位置的父目录中删去一个目录项，之后在新位置的父目录中新增一个目录项，然后注意修改inode的time。

- fs_write
  用Ptopi函数解析路径得到文件inode，用和fs_read相同的枚举方法得到需要新分配的块数，之后再逐块写即可。最后记得修改文件的inode。

- fs_truncate
  用Ptoi函数解析路径得该文件inode，然后计算修改前后所需的数据块数目，并分类讨论进行对应的处理。如果块数变多，则新分配相应数量的块；否则则删除相应数量的块，而大小不是 4kb 的整数倍时，最后一个块的情况需要格外注意。

- fs_utime、fs_statfs
  直接解析路径并修改或记录相应内容即可。

反思与总结

1. 最开始的一周其实对实验毫无思路，感觉建立如课本上的结构会很复杂，后来参考了前辈大牛Menci的思想，才发现可以把模型进行简化，构建起来也相对容易一些。

2. 删除数据块的实现，其实可以直接删除对数据的访问，但该系统直接将删除的数据抹除为0了，这一点可以尝试改进。

3. 分配块和删除块根据所求的块是一个还是多个进行了分类讨论，后续可以尝试将两个函数进行合并。