中国农业大学

操作系统实验

(2019 -2020 学年秋季学期)

实验名称：操作系统

任课教师：段青玲

班　　级：计算机 172

学　　号：2017304010413

姓　　名：张靖祥

时　　间：2019/12/31

实验 6 磁盘存储空间的管理

张靖祥 2017304010413 计算机
请将代码导入VS中，否则会出现中文乱码。

1. 前言——写在与课程相关内容之前

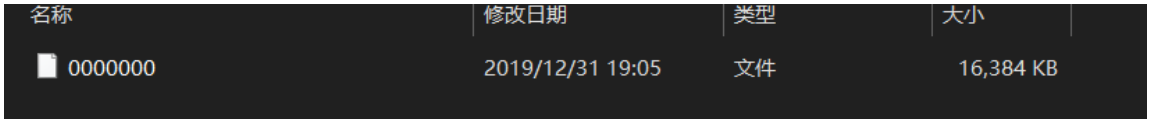
鉴于我对操作系统这门课，或者说这个内容有十分浓厚的兴趣，我决定做一个能模仿攀比ext2 文件系统的十分复杂且强大的文件系统。但最近实在事务繁重，先遇人工智能大作业加答辩和报告，又遇虚拟现实答辩大作业加答辩和报告，最后本周一（12 月 30 日）计算机组成原理大作业加答辩和报告，恐怕只能大体完成内容。然而上周日又突发肠胃不适，发烧 2 天，周一答辩完就昏倒在床上。今早（12 月 31 日）意识清醒后突然意识到 2020 年都要到了，我的操作系统还没写完呢，马上滚鞍下马（床）开始补救。已知程序bug的数量与程序规模成正比，改bug的难度与程序规模成二次方关系，单单一个ls指令就调了 2 个小时bug。最后，只能说将将实现了基本功能，虎头蛇尾了。

2. 数据结构与类的功能全解

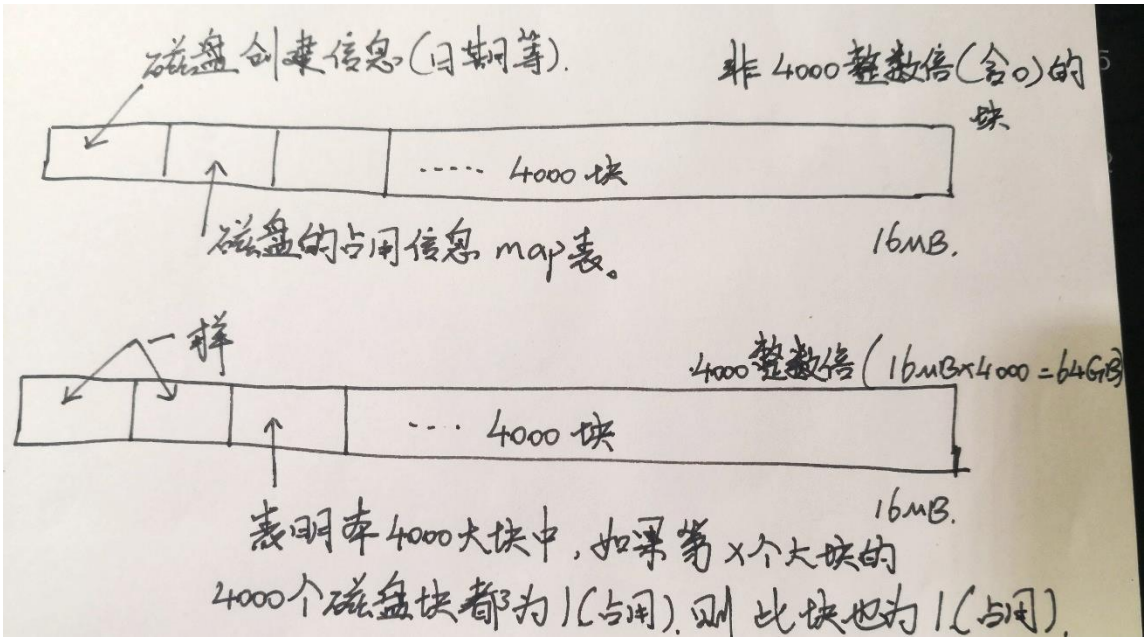
1 物理层Disk.h头文件

首先为了实现一个功能强大的文件系统，我的文件系统长度可以动态改变，占用的不是内存，而是真实的磁盘。数据结构如下：

每一个大磁盘块占用 16MB（16*1024=16384）的内存空间，当超出范围后会动态新建一个大磁盘块（如下图），删除后会动态减少一个大磁盘块。而程序的存储单位为小磁盘块（簇），4Kb。



在每一个大磁盘块中，有 4000 个簇，4kb一个簇。大磁盘块中的第一块固定为磁盘创建的基本信息（创建时间等），第二块固定为本磁盘块的占用情况（map表），前 512bk有效。当大磁盘块的编号为 0000000,0004000 等 4000 的整数倍时，第三块为本 4000 块大磁盘的占用情况，前 512kb有效。当一个大磁盘块占用了最后一个簇后，会向 4000 倍数的组大磁盘块添加本大磁盘块全都占满的信息。这样在查找空闲磁盘的时候，可以通过双重索引快速找到空闲磁盘块。



类函数：

```
private:
    DiskInformation __CreateDiskInformation(); //返回一个创建磁盘的信息
    int __FindEmptyFromMap(char* DiskMapBlock);  //通过读出的Map表找到空闲盘块
    char* __GetDiskName(int number); //将编号转化为路径
    int __SearchNotExistName();  //找一个磁盘块不存在的名字
public:
    DiskPhysical() {}
    char* ReadDisk(unsigned int block_num);  //根据盘块读一个内容
    void WriteDisk(unsigned int block_num, char* content);//写一个盘块，覆盖
    void CreateDisk();           //创建一个没有的盘块
    unsigned int SearchEmpty();  //注意这里返回的块号有32位，每块4kb，最多控制16Tb空间
    void AddToOccupy(unsigned int BlockNumber);  //将一个盘块增加占用
    void DelFromOccupy(unsigned int BlockNumber); //删掉一个盘块占用
    unsigned int GetLastDiskNum();
    DiskInformation getDiskInfo(int diskNum);
```

DiskPhysical类

私有函数：

__CreateDiskInformation（）//函数用来创建的磁盘基本信息，比如当前的时间。

int __FindEmptyFromMap(char* DiskMapBlock)//用来在一个 512kb的map中返回一个为 0 的索引

char* __GetDiskName(int number)// 通过编号返回磁盘位置字符串（12 返回disk:\0000012）

int __SearchNotExistName() //找到没有存在的大组磁盘的编号，用于磁盘占用满，添加新磁盘的时候找到可以命名的磁盘

共有函数：

char* ReadDisk(unsigned int block_num); //根据盘块号读一个内容

void WriteDisk(unsigned int block_num, char* content);//写一个盘块，覆盖

void CreateDisk();   //创建一个没有的盘块，内部调用__SearchNotExistName()

unsigned int SearchEmpty(); //注意这里返回的块号有 32 位，每块 4kb，最多控制 16Tb空间

void AddToOccupy(unsigned int BlockNumber); //将一个盘块增加占用

```
void DelFromOccupy(unsigned int BlockNumber); //删掉一个盘块占用
unsigned int GetLastDiskNum();  //等于__SearchNotExistName()
DiskInformation getDiskInfo(int diskNum);  //返回磁盘创建的信息
```

2 磁盘缓冲DiskBuffer.h头文件:

由于本程序中我需要对底层文件进行大量的读写,所以我计划在中间层实现一个磁盘缓冲的功能，但是由于时间关系，我这里仅仅做了一个Disk.h中DiskPhysical类的封装，没有实现基本功能（未来可以补充）

```
DiskB(){}
char* read(unsigned int block_num);  //根据盘块读一个内容
void write(unsigned int block_num, char* content);//写一个盘块，覆盖
unsigned int findEmpty();  //注意这里返回的块号有32位，每块4kb，最多控制16Tb空间
void addToOccupy(unsigned int BlockNumber);  //将一个盘块增加占用
void delFromOccupy(unsigned int BlockNumber); //删掉一个盘块占用
unsigned int getLastDiskNum();
DiskInformation getDiskInfo(int diskNum);
```

```
DiskInformation DiskB::getDiskInfo(int diskNum) {
    return DP.getDiskInfo(diskNum);
}


char* DiskB::read(unsigned int block_num) {
    return DP.ReadDisk(block_num);
}

void DiskB::write(unsigned int block_num, char* content) {
    DP.WriteDisk(block_num, content);
}
```

3 文件底层实现FCB.h头文件

本题我都fcb分为两个，一个是磁盘的真实的物理存储另一个是为了方便操作而定义的虚拟fcb，详细说明如下

```
struct ext2_inode_physical { //一个inode占用30个int的大小，即120个字节
    //单个块为4Kb，即可以存储40个块，需要6位进行块内inode寻址
    //此结构体表示的是物理存储结构
    unsigned int i_ABMS;      //这个32位int表示如下：
    //0~2位：文件模式，0为空，1为可目录，2位文本文件等等
    //3~11位：authority权限，与linux表示方法一样
    unsigned int i_uid;       //文件使用者的id
    unsigned int i_size;      //文件大小这里储存低32，单文件最大为4Gb
    unsigned int i_ctime;     //创建时间
    unsigned int i_mtime;     //修改日期
    unsigned int blocks;
    char i_name[44];
    //文件名，最长为38位，
    //40为是上个结点的孩子还是兄弟
    //41位为i_last_inter的8位
    //42位为i_son_inter的8位
    //43位为i_brother_inter的8位
    unsigned int i_block[10];  //指向物理块的指针，本系统最大采用2级索引，
    unsigned int i_son_outer;     //如果是目录，则表示子目录或文件的inode位置
    unsigned int i_brother_outer;//表示和在该目录下的兄弟结点的inode位置
    unsigned int i_last_outer;   //表示和在该目录下的兄弟结点的inode位置
};
```

每个fcb大小为 30 个int（120kb），所以每个簇（4000kb）中存 30 个fcb。

在这个结构体里，权限（rwe共 9 位），文件模式（3 位）共用 1 个int，称为 i_ABMS，时间用一个int，名字 40 位长度，41,42,43,44 另有他用（见上图）。之所以这样设置是为了减少空间占用。本系统采用孩子兄弟链表表示文件的链接，故需要两个指针，另有一个存储父亲的指针。但是每个指针分为两部分一部分是簇编号（int类型），一部分是簇内编号（char类型），最大为 30。

Blocks 为二级索引，10 个指针，所以最大单个文件的大小为 8*4096+1024*4096+1024*1024*4096b

```
struct i_Time {
    int YY;
    int MM;
    int DD;
    int hh;
    int mm;
};
```

```
struct i_Authority {
    bool Read_Me;
    bool Read_Other;
    bool Read_Visitor;
    bool Write_Me;
    bool Write_Other;
    bool Write_Visitor;
    bool Execute_Me;
    bool Execute_Other;
    bool Execute_Visitor;
};
```

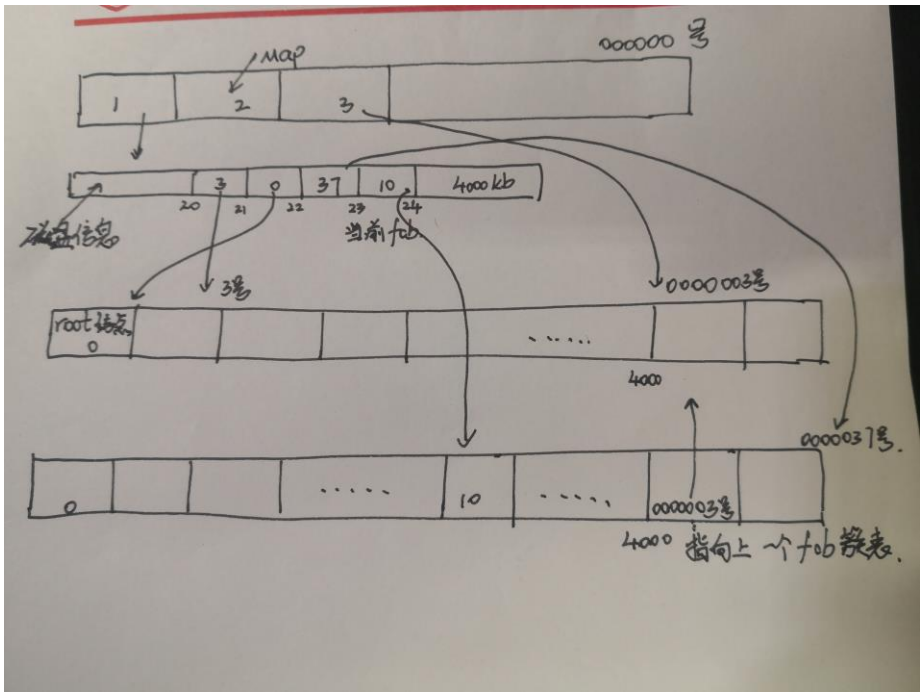
```
enum i_FileMode {
    Empty=0,
    Content,
    File,
    Exe,
    Link,
    Equipment
};
```

在磁盘块 0000000 的第 1 个簇的 20 号开始记录了 4 个信息，如下

```
struct RootInfo {
    unsigned int son_num;
    unsigned int son_offset;
    unsigned int current_block_inode_number;
    unsigned int current_block_location;
};
```

前两个表明root根目录fcb对应的位置，current_block_inode_number表明当前写入的fcb表的个数，current_block_location为fcb簇位置。新建fcb的时候会在current_block_location的盘块建一个fcb，并且current_block_inode_number++，如果current_block_inode_number大于 30，会重新找一个空的盘块用于记录fcb。同时在新建立的current_block_inode的第 4000 个位置记录一个指向之前的fcb簇的指针。

当删除一个fcb的时候，除了删掉它对应的所有的blocks中的对应的内容，还要将最后一个fcb移动到被删掉的那个fcb的位置。需要修改最后一个fcb父亲或左兄弟的指针，以及被删掉的那个fcb的父亲或左兄弟的指针。



而用于方便操作的是这个结构体

```
struct ext2_inode { //此结构体表示的是逻辑结构，需要进行转换
    i_FileMode i_mode;//文件模式，0为空，1为可目录，2位文本文件等等
    i_Authority i_authority;//authority权限，与linux表示方法一样
    unsigned int i_uid;        //文件使用者的id
    unsigned int i_size;       //文件大小这里储存低32，单文件最大为4Gb
    i_Time i_ctime;      //创建时间
    i_Time i_mtime;      //修改日期
    char i_name[43];  //文件名，最长为37位
    unsigned int i_blocks;//block数量，block数量上限为：8个物理块
    unsigned int i_block[10];  //指向物理块的指针，本系统最大采用2级索引，
    unsigned int i_son_outer;     //如果是目录，则表示子目录或文件的块位置
    unsigned int i_son_inner;     //目录情况下，inode结点的块内位置
    unsigned int i_brother_outer;  //表示和在该目录下的兄弟结点的inode位置
    unsigned int i_brother_inner;  //目录情况下，inode结点的块内位置，每个块内可以存储40个
    unsigned int i_last_outer;  //表示和在该目录下的兄弟结点的inode位置
    unsigned int i_last_inner;  //目录情况下，inode结点的块内位置，每个块内可以存储40个
    int is_last_father;          //1表示上个结点为父节点，0为兄弟结点
};
```

与上述内容相似，不再赘述

类的成员函数如下

```
private:
    ext2_inode *ext2;
    int last_read_location;
    int last_read_offset;
    int content_location;
    DiskB disk;
    int self_location;
    int self_offset;

    void __setAuthority(int* i_ABS, int authority);
    i_Authority __getAuthority(int i_ABS);
    int __getAuthority(i_Authority authority);
    void __setImode(int* i_ABS, i_FileMode mode);
    i_FileMode __getMode(int i_ABS);
    i_Time __getTimeNow();
    int __ConvertTimeToInt(i_Time time);
    i_Time __ConvertIntToItime(int time);
    void __FCBPlus();
    void __FCBMinus();

    int __getBlocks();
    void __setChangeTime();
```

私有函数：

void __setAuthority(int* i_ABS, int authority);//用于int权限表示转化为i_ABS表示

i_Authority __getAuthority(int i_ABS); //从i_ABS权限转为int表示

int __getAuthority(i_Authority authority); //从i_Authority转为int表示

void __setImode(int* i_ABS, i_FileMode mode);// 用于int文件模式表示转化为i_ABS表示

i_FileMode __getMode(int i_ABS); //从i_ABS文件模式转为i_FileMode表示

i_Time __getTimeNow(); //返回当前时间

int __ConvertTimeToInt(i_Time time); //时间转为int表示

i_Time __ConvertIntToItime(int time); //int转为i_Time

void __FCBPlus(); //FCB表数量加 1

void __FCBMinus();//FCB表数量减 1

int __getBlocks(); //获取当前文件占用磁盘的块数

void __setChangeTime(); //将当前时间写入fcb文件改变时间

```
public:
    Inode();  //创建一个空Inode类，用于调用函数
    Inode(int block_num, int offset);  //由给定的Inode结点导入
    ~Inode();

    void createRoot();
    RootInfo getRoot(); //返回根节点
    void createFCB(int father_num,int father_offset,bool isson,
        int* create_num, int* create_offset, i_FileMode fileMode, int uid,
        const char *name, int authority);

    ext2_inode* getFCB(); //获取一个inode信息
    i_FileMode getMode();
    i_Authority geteAuthority();
    int getAuthroity();
    unsigned int getUid();
    unsigned int getSize();
    i_Time getCreateTime(); //返回Inode中的时间
    i_Time getChangeTime();
    char* getName();

    void setImode(i_FileMode fileMode);
    void setAuthority(int authority);
    void setName(char* name);
```

```
    void setImode(i_FileMode fileMode);
    void setAuthority(int authority);
    void setName(char* name);

    void getSon(int* num, int* offset);
    void getBrother(int* num, int* offset);
    void getFather(int* num, int* offset, int *isSon);
    void getNextSon(int* num, int* offset);
    void getLastSon(int* num, int* offset);
    void setBrother(int num, int offset);
    void setSon(int num, int offset);
    char* getNextContent(int* length);
    void writeContent(char* content, int length);

    void delContent();
    void delFCB();
    void saveFCB();
    void showInTerminal();
```

公有函数：

（太多了今天没时间写完了，老师请见谅）见名知意吧

4 文件操作实现filestream.h头文件

这个没有什么数据结构了，就是fcb的封装

```
#define ROOT 0
#define VISITOR 60000
```

Root用户为0，visitor用户的uid大于60000

用ls指令返回的文件列表信息如下结构

```
struct fileInfo {
    char* name;
    int authority;
    int uid;
    int size;
    i_FileMode mode;
    i_Time ctime;
    i_Time mtime;
    int blocknum;
    int blockoffset;
    int sonnum;
    int sonoffset;
};
```

文件打开方式（实现了），指针移动方式（没有实现此功能）

```
enum fileOpenMode {
    readmode = 1,
    writemode = 2,
    exemode = 4,
    create = 8
};
```

```
enum seek_mode{
    beg = 1,
    cur = 2,
    end = 3
};
```

文件类的私有成员

```
class FileStream {
private:
    fileInfo FileRoute[100];
    int now_depth;
    int uid;
    Inode fcb;

    unsigned int lastSonLocation;
    unsigned int lastSonOffset;

    int openFileLocation;
    int openFileOffset;
    char writeBuffer[4096];
    int currentSize;
    int filePointer;
    int fileSize;
    fileOpenMode openMode;
    i_FileMode fileMode;
    progress currentPorgress[100];

    void __delFile(unsigned int num, unsigned int offset);
    int __findEmptyProgress(fileOpenMode openMode, unsigned int PCBlocation, unsigned int PCBoffset);
```

共有函数

```
public:
    FileStream(int userid);
    FileStream();
    fileInfo getRoot();

    int formatDisk();
    int getBatchSon(fileInfo* info, int number);
    int getBatchSon(int num, int offset, fileInfo* info, int length);
    void getPWD();
    int cdDirectory(const char* name);
    void setUid(int uid);
    int getUid();
    void depthDeduce();
    void depthToOne();
    void getNodeByName(const char* name, unsigned int* num, unsigned int* offset, i_FileMode mode);
    void showopen();
```

```
    int mk(char* name, int aut,i_FileMode mode);
    void openFile(char* name, fileOpenMode openMode);
    void closeFile(char* name);
    void writeFile(char* name, char* content, int length);
    char* readFile(char* name,int *length);
    void flush();
    void seekg(int pos, seek_mode mode);
    void seekp(int pos, seek_mode mode);
    void read(char *content, int length);

    void delFile(char* name);
    void moveFile(char* From, char* To);
};
```

没时间解释了，望老师见名知意。后面几个标下划线的是没有实现的函数

5 指令解释CMD.h头文件

没什么数据结构需要说明，但是bug到挺难调试的，如下

```
class Command {
private:
    FileStream filestream;
    void __changeColor(i_FileMode mode);
    int uid;
public:
    Command();
    int login();
    int execute(char *command);
    void showPWD();
    int list(const char* method);
    int mkdir(char *name, const char *authority);
    int cd(const char *name);
    int del(const char* name);
    int open(const char*mode, const char* name);
    int write(const char* name, const char* content);
    int close(const char* name);
    int read(const char* name);
    int showopen();
};
```

6 外部工具tools.h头文件

只有一个分割split函数

```
void split(char* src, const char* separator, char** dest, int* num) {
    char* pNext;
    int count = 0;
    if (src == NULL || strlen(src) == 0)
        return;
    if (separator == NULL || strlen(separator) == 0)
        return;
    pNext = strtok(src, separator);
    while (pNext != NULL) {
        *dest++ = pNext;
        ++count;
        pNext = strtok(NULL, separator);
    }
    *num = count;
}
```

7 main函数

```
printf("Providence Azure [版本 1.0.001]\n");
printf("(c) 2019 计算机172，张靖祥 Inc。保留所有权利。\n");
Command cmd;
string com;

while (true) {
    if (cmd.login() == 1)
        break;
}

while (true) {
    cmd.showPWD();
    getline(cin, com);
    int result = cmd.execute((char*)com.c_str());
    if (result == 0) break;
}
```

3. 演示可用指令

登录目前只能root，其他的登录可以后续补充

```
Providence Azure [版本 1.0.001]
(c) 2019 计算机172，张靖祥 Inc。保留所有权利。
account: root
pwd:

login successful. user: root
```

首次登录一定要初始化磁盘

```
root:>formatdisk
```

```
login successful. user: root
root:>formatdisk
系统找不到指定的文件。
root:>ls
```

出现这个提示为正常现象

```
root:>mkdir test
root:>mkdir test2
root:>ls
          test          test2

root:>cd test
root:\test>mkdir test3
root:\test>l
l is not a vaild command.
root:\test>ls
          test3

root:\test>cd ..
root:>open create me.txt
root:>ls -l
        name  uid  authroity     size   mode    createtime          modifytime
      test     0      777          0     1    2019/12/31 20:42   2019/12/31 20:42
      test     0      777          0     1    2019/12/31 20:42   2019/12/31 20:42
      test     0      777          0     1    2019/12/31 20:42   2019/12/31 20:42
```

```
root:>close me.txt
root:>write me.txt providence
file not open.
root:>open write me.txt
root:>write me.txt providence
root:>read me.txt
providence
```

嗯，刚刚好像ls –l的功能出现了bug，改了下重进是这样的

```
Microsoft Visual Studio 调试控制台
Providence Azure [版本 1.0.001]
(c) 2019 计算机172，张靖祥 Inc。保留所有权利。
account: root
pwd:

login successful. user: root
root:>ls -l
        name  uid  authroity     size   mode    createtime          modifytime
      test     0      777          0     1    2019/12/31 20:42   2019/12/31 20:42
     test2     0      777          0     1    2019/12/31 20:42   2019/12/31 20:42
    me.txt     0      777         10     2    2019/12/31 20:43   2019/12/31 20:43
        ss     0      777          0     1    2019/12/31 20:45   2019/12/31 20:45


root:>quit

F:\大三上\操作系统实验\实验6 文件管理2\MySystem\Debug\ConsoleApplication1.exe (进程 23028)已退出，
按任意键关闭此窗口...
```

4. 有待完善的功能

还有很多可以完善的地方

5. 心得体会

本实验中，我十分充分的学习到了磁盘空闲空间的记录方法（空闲表法，空闲链表法，位示图（本实验我所使用的），成组链接法），文件的物理