操作系统课程设计

主 要 内 容

1 题目

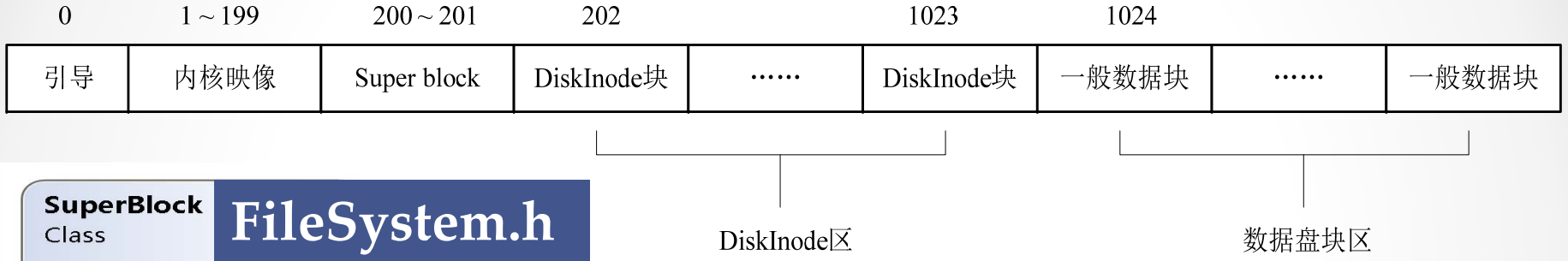
2 设计报告要求

3 提交时间

4 答疑

一、UNIX V6++文件系统代码分析

1. 磁盘文件结构



SuperBlock Class — **FileSystem.h**

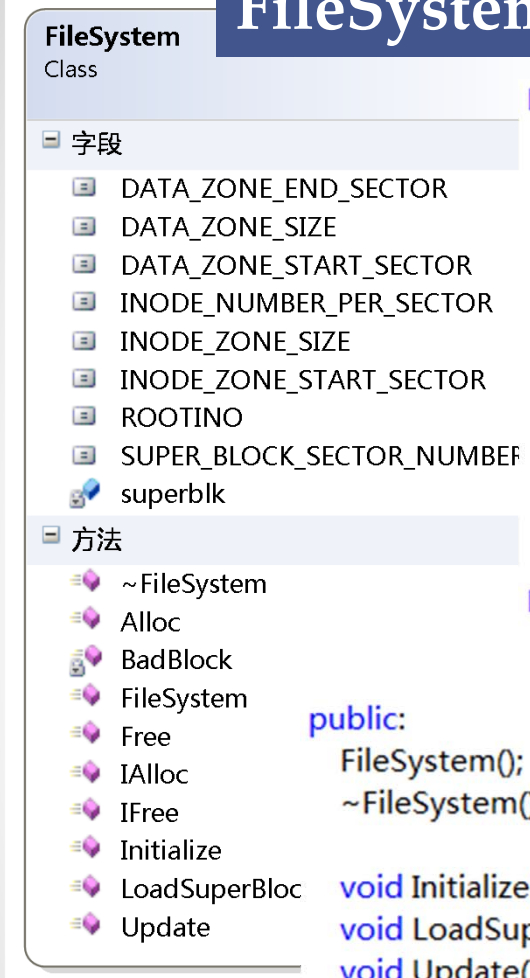
- 字段
  - padding
  - s_flock
  - s_fmod
  - s_free
  - s_fsize
  - s_ilock
  - s_inode
  - s_isize
  - s_nfree
  - s_ninode
  - s_ronly
  - s_time
- 方法
  - ~SuperBlock
  - SuperBlock

DiskInode Class — **Inode.h**

- 字段
  - d_addr
  - d_atime
  - d_gid
  - d_mode
  - d_mtime
  - d_nlink
  - d_size
  - d_uid
- 方法
  - ~DiskInode
  - DiskInode

按SuperBlock和DiskInode类进行数据解析

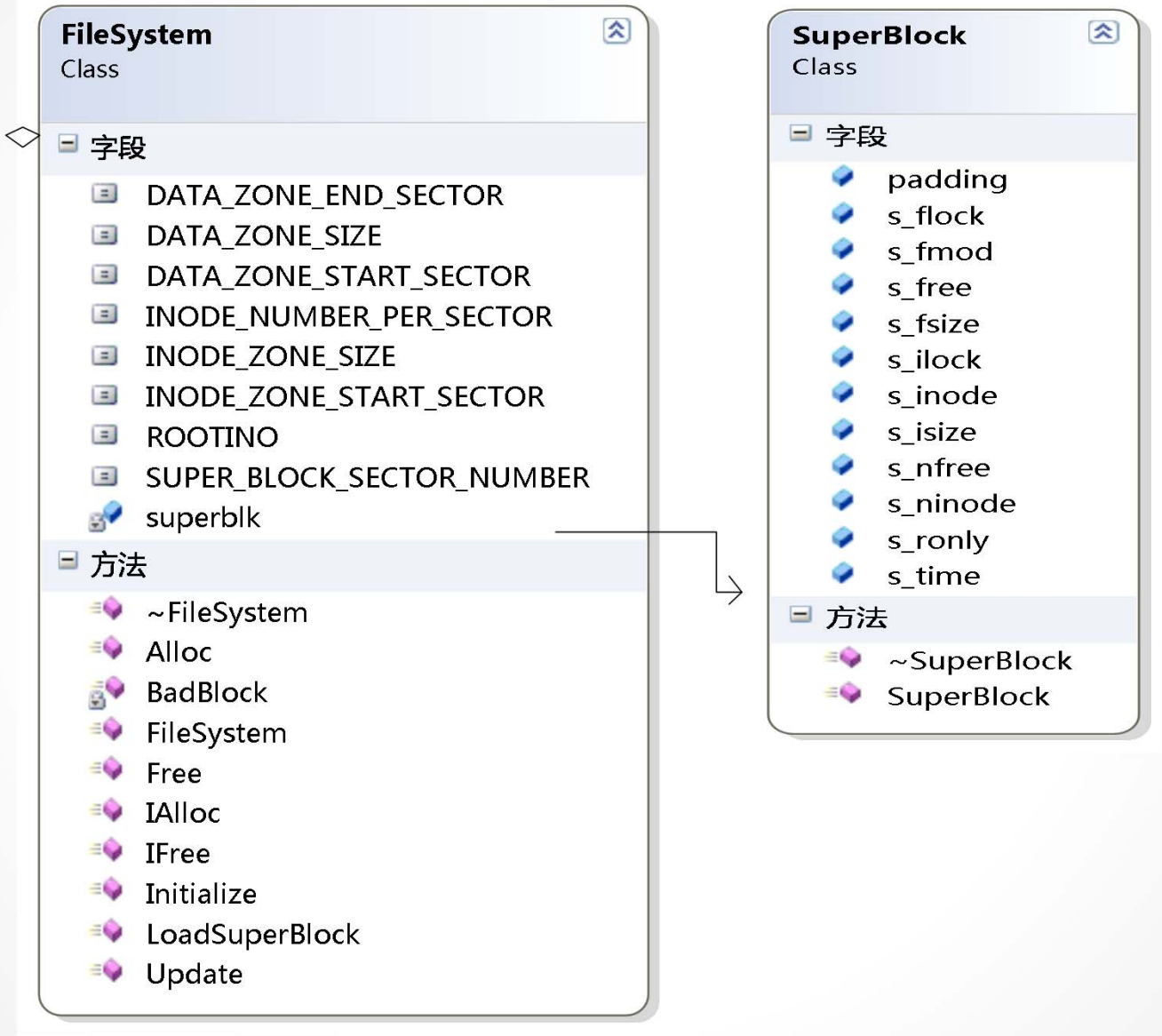
FileSystem.h

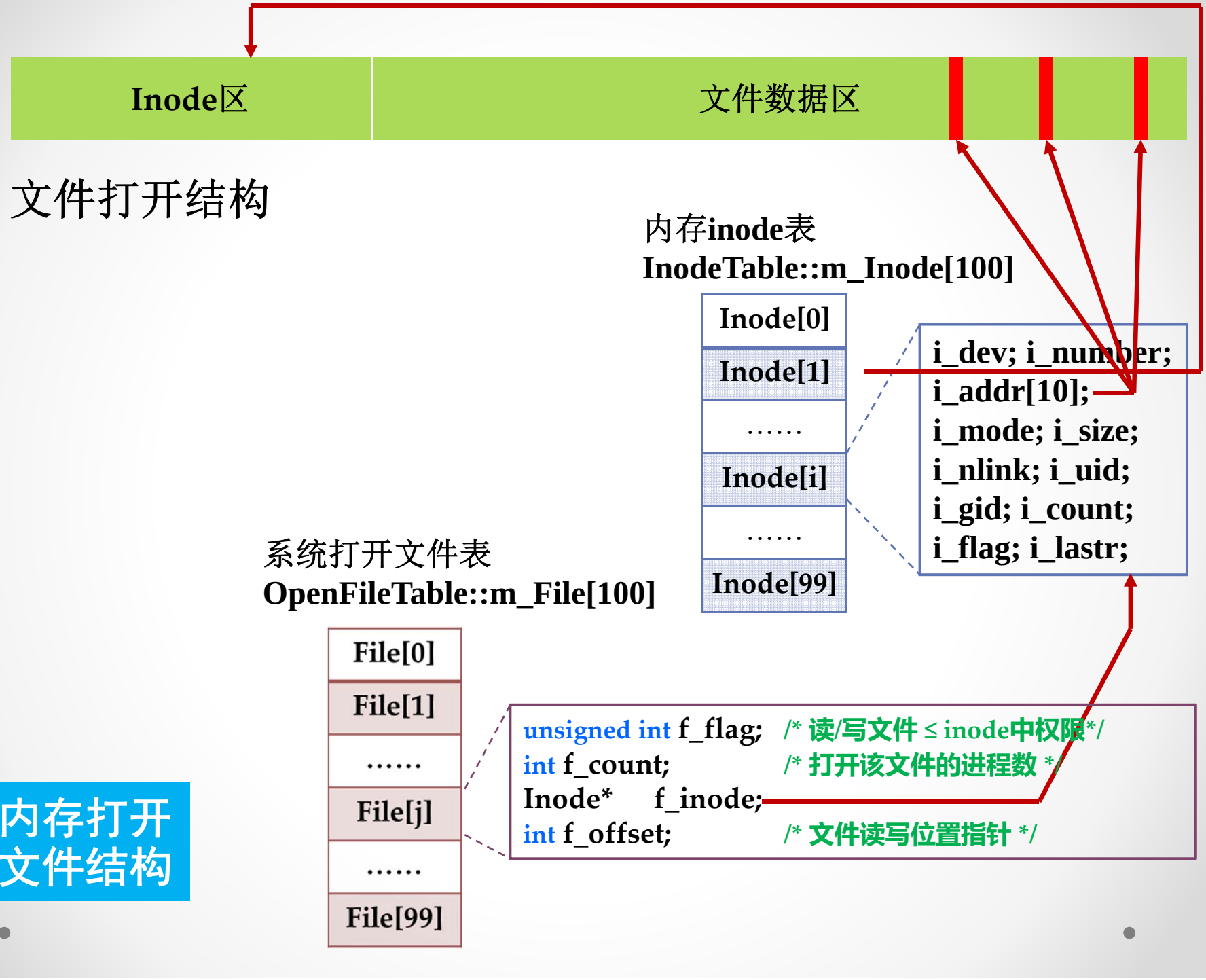



```
public:
    /* static consts */
    static const int SUPER_BLOCK_SECTOR_NUMBER = 200;   /* 定义SuperBlock位于磁盘上的扇区号，
    static const int ROOTINO = 0;       /* 文件系统根目录外存Inode编号 */
    static const int INODE_NUMBER_PER_SECTOR = 8;     /* 外存INode对象长度为64字节，每个磁盘块可
    static const int INODE_ZONE_START_SECTOR = 202;     /* 外存Inode区位于磁盘上的起始扇区号 */
    static const int INODE_ZONE_SIZE = 1024 - 202;     /* 磁盘上外存Inode区占据的扇区数 */

    static const int DATA_ZONE_START_SECTOR = 1024;     /* 数据区的起始扇区号 */
    static const int DATA_ZONE_END_SECTOR = 18000 - 1; /* 数据区的结束扇区号 */
    static const int DATA_ZONE_SIZE = 18000 - DATA_ZONE_START_SECTOR;   /* 数据区占据的扇区数

    /* Members */
private:
    SuperBlock superblk;     /* 内存中的SuperBlock副本 */
```

FileSystem.cpp中定义的成员函数

```
public:
    FileSystem();
    ~FileSystem();

    void Initialize();          /* 初始化成员变量 */
    void LoadSuperBlock();      /* 系统初始化时读入SuperBlock */
    void Update();              /* 将SuperBlock对象的内存副本更新到存储设备的SuperBlock中去 */
    Inode* IAlloc();            /* 分配一个空闲外存INode，一般用于创建新的文件 */
    void IFree(int number);     /* 释放编号为number的外存INode，一般用于删除文件 */
    int Alloc(short dev);       /* 分配空闲磁盘块 */
    void Free(int blkno);       /* 释放编号为blkno的磁盘块 */

private:
    bool BadBlock(SuperBlock* spb, short dev, int blkno);   /* 检查编号blkno的磁盘块是否属于数据盘块区 */
```


FileSystem
Class
字段
DATA_ZONE_END_SECTOR
DATA_ZONE_SIZE
DATA_ZONE_START_SECTOR
INODE_NUMBER_PER_SECTOR
INODE_ZONE_SIZE
INODE_ZONE_START_SECTOR
ROOTINO
SUPER_BLOCK_SECTOR_NUMBER
superblk
方法
~FileSystem
Alloc
BadBlock
FileSystem
Free
IAlloc
IFree
Initialize
LoadSuperBlock
Update
SuperBlock
Class
字段
padding
s_flock
s_fmod
s_free
s_fsize
s_ilock
s_inode
s_isize
s_nfree
s_ninode
s_ronly
s_time
方法
~SuperBlock
SuperBlock


Inode区
文件数据区
2. 文件打开结构
内存inode表
InodeTable::m_Inode[100]
Inode[0]
Inode[1]
……
Inode[i]
……
Inode[99]
i_dev; i_number;
i_addr[10];
i_mode; i_size;
i_nlink; i_uid;
i_gid; i_count;
i_flag; i_lastr;
系统打开文件表
OpenFileTable::m_File[100]
File[0]
File[1]
……
File[j]
……
File[99]
unsigned int f_flag; /* 读/写文件≤inode中权限*/
int f_count; /* 打开该文件的进程数 */
Inode* f_inode;
int f_offset; /* 文件读写位置指针 */
内存打开
文件结构

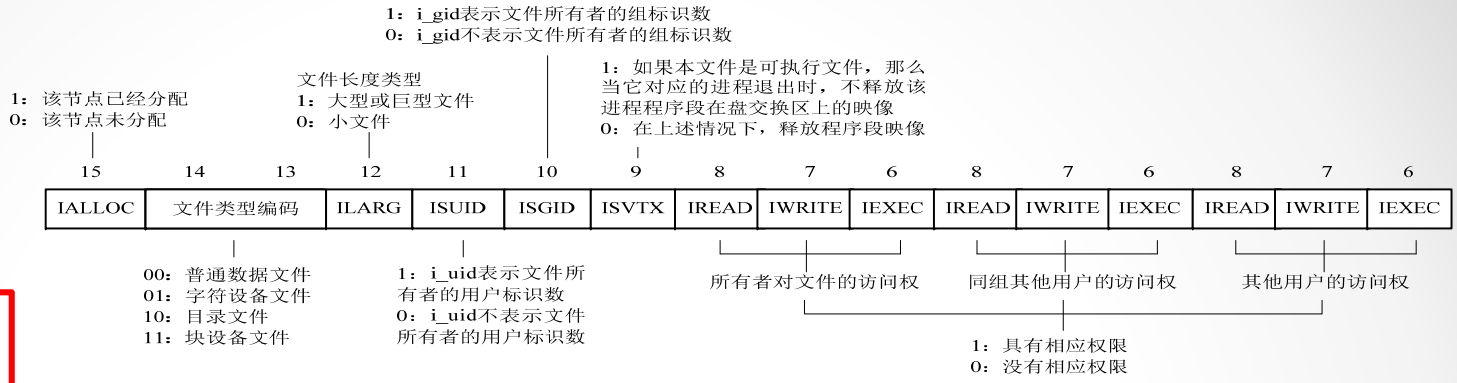
Inode.h

Inode
Class

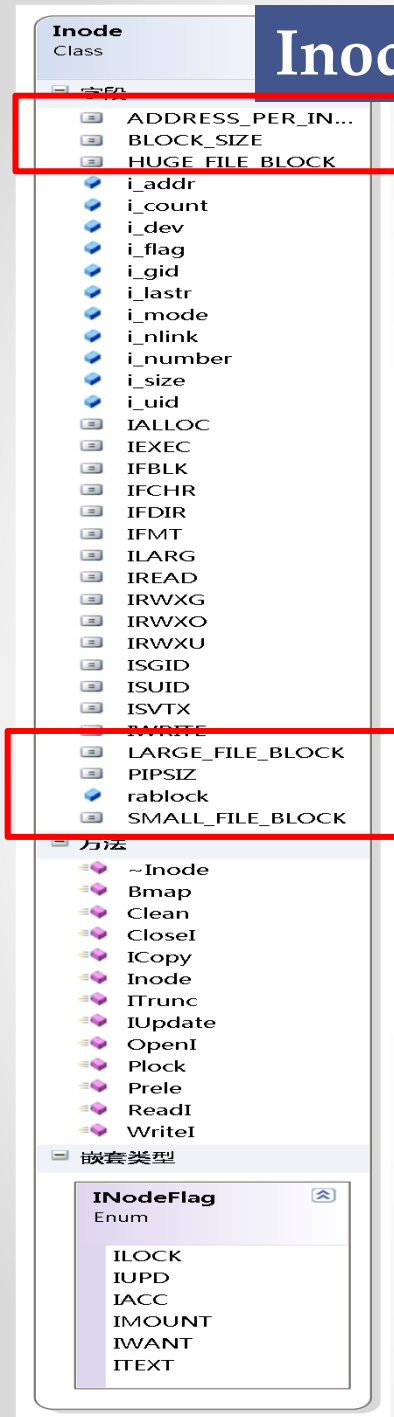
- 字段
  - ADDRESS_PER_IN...
  - BLOCK_SIZE
  - HUGE_FILE_BLOCK
  - i_addr
  - i_count
  - i_dev
  - i_flag
  - i_gid
  - i_lastr
  - i_mode
  - i_nlink
  - i_number
  - i_size
  - i_uid
  - IALLOC
  - IEXEC
  - IFBLK
  - IFCHR
  - IFDIR
  - IFMT
  - ILARG
  - IREAD
  - IRWXG
  - IRWXO
  - IRWXU
  - ISGID
  - ISUID
  - ISVTX
  - IWRITE
  - LARGE_FILE_BLOCK
  - PIPSIZ
  - rablock
  - SMALL_FILE_BLOCK
- 方法
  - ~Inode
  - Bmap
  - Clean
  - CloseI
  - ICopy
  - Inode
  - ITrunc
  - IUpdate
  - OpenI
  - Plock
  - Prele
  - ReadI
  - WriteI
- 嵌套类型

INodeFlag
Enum

- ILOCK
- IUPD
- IACC
- IMOUNT
- IWANT
- ITEXT




```
/* static const member */
static const unsigned int IALLOC = 0x8000;     /* 文件被使用 */
static const unsigned int IFMT = 0x6000;      /* 文件类型掩码 */
static const unsigned int IFDIR = 0x4000;     /* 文件类型：目录文件 */
static const unsigned int IFCHR = 0x2000;     /* 字符设备特殊类型文件 */
static const unsigned int IFBLK = 0x6000;     /* 块设备特殊类型文件，为0表示常规数据文件 */
static const unsigned int ILARG = 0x1000;     /* 文件长度类型：大型或巨型文件 */
static const unsigned int ISUID = 0x800;    /* 执行时文件时将用户的有效用户ID修改为文件所有者的User ID */
static const unsigned int ISGID = 0x400;    /* 执行时文件时将用户的有效组ID修改为文件所有者的Group ID */
static const unsigned int ISVTX = 0x200;      /* 使用后仍然位于交换区上的正文段 */
static const unsigned int IREAD = 0x100;      /* 对文件的读权限 */
static const unsigned int IWRITE = 0x80;      /* 对文件的写权限 */
static const unsigned int IEXEC = 0x40;       /* 对文件的执行权限 */
static const unsigned int IRWXU = (IREAD|IWRITE|IEXEC);     /* 文件主对文件的读、写、执行权限 */
static const unsigned int IRWXG = ((IRWXU) >> 3);       /* 文件主同组用户对文件的读、写、执行权限 */
static const unsigned int IRWXO = ((IRWXU) >> 6);       /* 其他用户对文件的读、写、执行权限 */
```



Inode.h

```
static const int BLOCK_SIZE = 512;     /* 文件逻辑块大小: 512字节 */
static const int ADDRESS_PER_INDEX_BLOCK = BLOCK_SIZE / sizeof(int);   /* 每个间接索引表(或索引块)包含的物理盘块号 */

static const int SMALL_FILE_BLOCK = 6; /* 小型文件：直接索引表最多可寻址的逻辑块号 */
static const int LARGE_FILE_BLOCK = 128 * 2 + 6;/* 大型文件：经一次间接索引表最多可寻址的逻辑块号 */
static const int HUGE_FILE_BLOCK = 128 * 128 * 2 + 128 * 2 + 6; /* 巨型文件：经二次间接索引最大可寻址文件逻辑块号 */

static const int PIPSIZ = SMALL_FILE_BLOCK * BLOCK_SIZE;

/* static member */
static int rablock;    /* 顺序读时，使用预读技术读入文件的下一字符块，rablock记录了下一逻辑块号
                          经过bmap转换得到的物理盘块号。将rablock作为静态变量的原因：调用一次bmap的开销
                          对当前块和预读块的逻辑块号进行转换，bmap返回当前块的物理盘块号，并且将预读块
                          的物理盘块号保存在rablock中。 */

 /* Functions */
public:

 Inode();                    /* Constructors */
 ~Inode();                   /* Destructors */

 void ReadI();               /* 根据Inode对象中的物理磁盘块索引表，读取相应的文件数据 */
 void WriteI();              /* 根据Inode对象中的物理磁盘块索引表，将数据写入文件      */
 int Bmap(int lbn);          /* 将文件的逻辑块号转换成对应的物理盘块号 */
 void IUpdate(int time);     /* 更新外存Inode的最后的访问时间、修改时间 */
 void ITrunc();              /* 释放Inode对应文件占用的磁盘块 */
 void Clean();               /* 清空Inode对象中的数据   */
 void ICopy(Buf* bp, int inumber);   /* 将包含外存Inode字符块中信息拷贝到内存Inode中   */
 void Prele();               /* 对Inode解锁，并且唤醒因等待锁而睡眠的进程 */
```

Inode.cpp中定义的成员函数

OpenFileManager.h

```
  /* Functions */
public:
  InodeTable();          /* Constructors */
  ~InodeTable();         /* Destructors */

  void Initialize();                  /* 初始化对g_FileSystem对象的引用  */
  void UpdateInodeTable();    /* 将所有被修改过的内存Inode更新到对应外存Inode中   */
  Inode* GetFreeInode();     /* 在内存Inode表中寻找一个空闲的内存Inode  */
  /*
   * 根据指定设备号dev，外存Inode编号获取对应Inode。如果该Inode已经在内存中，对其
   * 上锁并返回该内存Inode，如果不在内存中，则将其读入内存后上锁并返回该内存Inode   */
  Inode* IGet(short dev, int inumber);
  /*
   * 减少该内存Inode的引用计数，如果此Inode已经没有目录项指向它，
   * 且无进程引用该Inode，则释放此文件占用的磁盘块。 */
  void IPut(Inode* pNode);
  /* 检查设备dev上编号为inumber的外存inode是否有内存拷贝，
   * 如果有则返回该内存Inode在内存Inode表中的索引   */
  int IsLoaded(short dev, int inumber);
```

```
  /* static consts */
public:
  static const int NINODE  = 100;/* 内存Inode的数量 */

    /* Members */
public:
  Inode m_Inode[NINODE];    /* 内存Inode数组，每个打开文件都会占用一个内存Inode */
  FileSystem* m_FileSystem;   /* 对全局对象g_FileSystem的引用 */
```

成员函数定义在OpenFileManager.cpp中

File.h

```
public:
    /* Enumerate */
    enum FileFlags
    {
        FREAD = 0x1,        /* 读请求类型 */
        FWRITE = 0x2,       /* 写请求类型 */
        FPIPE = 0x4         /* 管道类型 */
    };

    /* Functions */
public:
    File();         /* Constructors */
    ~File();        /* Destructors */

    /* Member */
    unsigned int f_flag;    /* 对打开文件的读、写操作要求 */
    int      f_count;       /* 当前引用该文件控制块的进程数量 */
    Inode*   f_inode;       /* 指向打开文件的内存Inode指针 */
    int      f_offset;      /* 文件读写位置指针 */
```

File.h

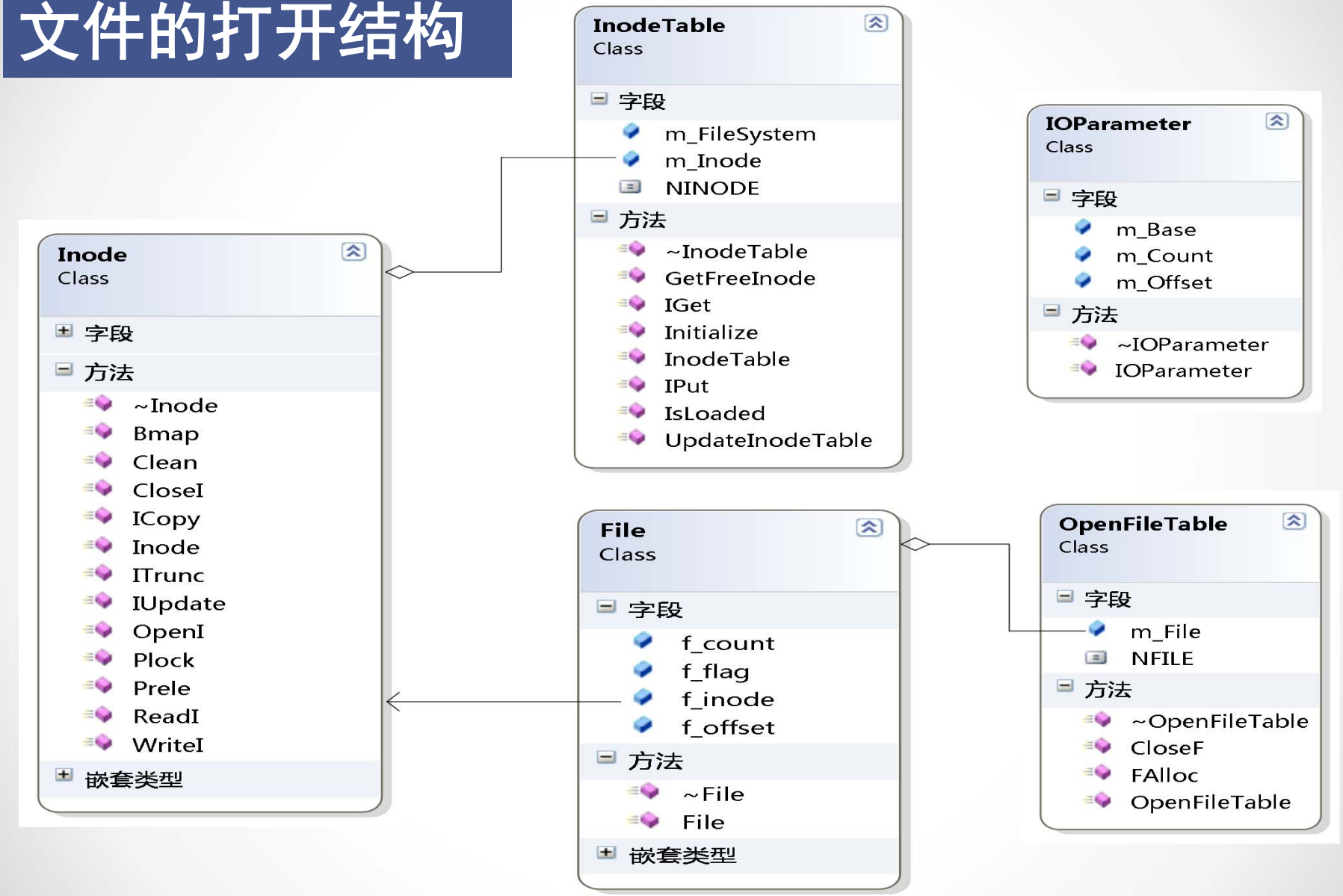
```
    /* Functions */
public:
    IOParameter();
    ~IOParameter();

    /* Members */
public:
    unsigned char* m_Base; /* 当前读、写用户目标区域的首地址 */
    int m_Offset;          /* 当前读、写文件的字节偏移量 */
    int m_Count;           /* 当前还剩余的读、写字节数量 */
```

成员函数定义在OpenFileManager.cpp中

OpenFileManager.h

```
public:
   /* static consts */
   static const int NFILE  = 100;/* 打开文件控制块File结构的数量 */

   /* Members */
public:
   File m_File[NFILE];        /* 系统打开文件表，为所有进程共享，进程打开文件描述符表
                              中包含指向打开文件表中对应File结构的指针。*/

   /* Functions */
public:
   OpenFileTable();     /* Constructors */
   ~OpenFileTable();    /* Destructors */

   File* FAlloc();          /* 在系统打开文件表中分配一个空闲的File结构   */
   /* 对打开文件控制块File结构的引用计数f_count减1，
    * 若引用计数f_count为0，则释放File结构。   */
   void CloseF(File* pFile);
```

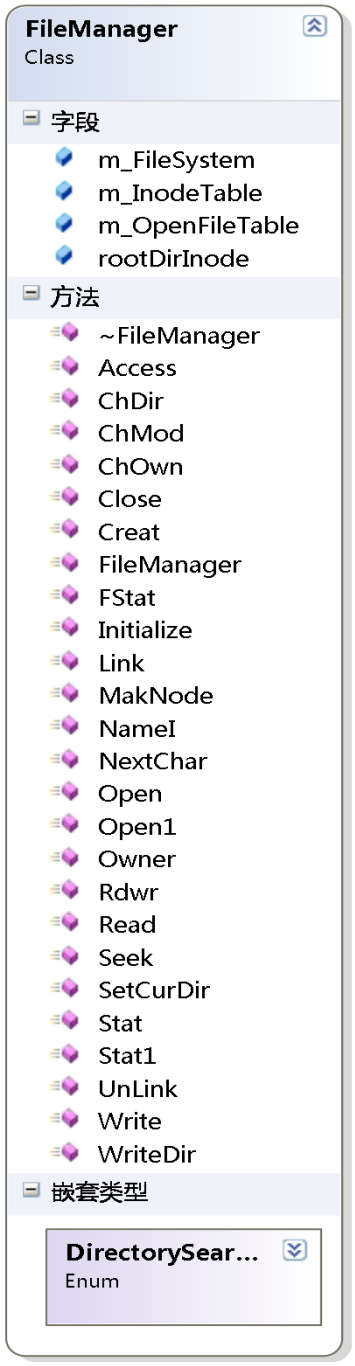
文件的打开结构


InodeTable
Class
字段
m_FileSystem
m_Inode
NINODE
方法
~InodeTable
GetFreeInode
IGet
Initialize
InodeTable
IPut
IsLoaded
UpdateInodeTable
IOParameter
Class
字段
m_Base
m_Count
m_Offset
方法
~IOParameter
IOParameter
Inode
Class
字段
方法
~Inode
Bmap
Clean
CloseI
ICopy
Inode
ITrunc
IUpdate
OpenI
Plock
Prele
ReadI
WriteI
嵌套类型
File
Class
字段
f_count
f_flag
f_inode
f_offset
方法
~File
File
嵌套类型
OpenFileTable
Class
字段
m_File
NFILE
方法
~OpenFileTable
CloseF
FAlloc
OpenFileTable

3. 文件目录结构

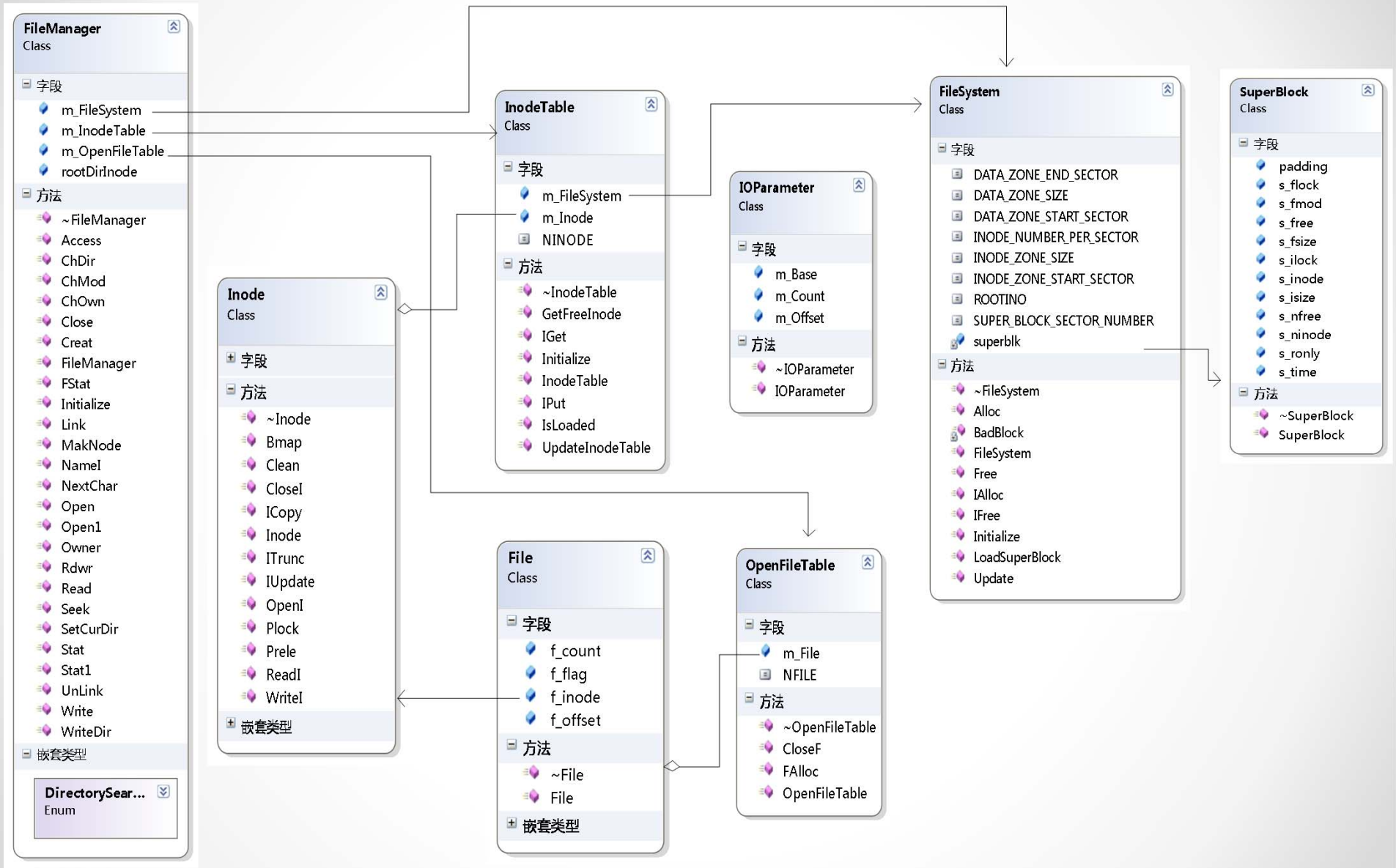

```
  /* static members */
public:
  static const int DIRSIZ = 28;     /* 目录项中路径部分的最大字符串长度 */

  /* Functions */
public:
  DirectoryEntry();
  ~DirectoryEntry();

  /* Members */
public:
  int m_ino;                        /* 目录项中Inode编号部分 */
  char m_name[DIRSIZ];              /* 目录项中路径名部分 */
```

4. 文件操作接口




```
  /* Functions */
public:
  FileManager();
  ~FileManager();

  void Initialize();                              /* 初始化对全局对象的引用 */
  void Open();                                    /* Open()系统调用处理过程 */
  void Creat();                                   /* Creat()系统调用处理过程 */
  void Open1(Inode* pInode, int mode, int trf);   /* Open()、Creat()系统调用的公共部分 */
  void Close();                                   /* Close()系统调用处理过程 */
  void Seek();                                    /* Seek()系统调用处理过程 */
  void FStat();                                   /* FStat()获取文件信息 */
  void Stat();                                    /* Stat()获取文件信息 */
  void Stat1(Inode* pInode, unsigned long statBuf);  /* FStat()和Stat()系统调用的共享例程 */
  void Read();                                    /* Read()系统调用处理过程 */
  void Write();                                   /* Write()系统调用处理过程 */
  void Rdwr(enum File::FileFlags mode);           /* 读写系统调用公共部分代码 */
  /* 目录搜索，将路径转化为相应的Inode，返回上锁后的Inode */
  Inode* NameI(char (*func)(), enum DirectorySearchMode mode);
  static char NextChar();                         /* 获取路径中的下一个字符 */
  Inode* MakNode(unsigned int mode);              /* 被Creat()系统调用使用，用于为创建新文件分配内核资源 */
  void WriteDir(Inode* pInode);                   /* 向父目录的目录文件写入一个目录项 */
  void SetCurDir(char* pathname);                 /* 设置当前工作路径 */
  int Access(Inode* pInode, unsigned int mode);/* 检查对文件或目录的搜索、访问权限，作为系统调用的辅助函数 */
  Inode* Owner();                                 /* 检查文件是否属于当前用户 */
  void ChMod();                                   /* 改变文件访问模式 */
  void ChOwn();                                   /* 改变文件文件所有者user ID及其group ID */
  void ChDir();                                   /* 改变当前工作目录 */
  void Link();                                    /* 创建文件的异名引用 *
  void UnLink();                                  /* 取消文件 */

public:
  Inode* rootDirInode;        /* 根目录内存Inode */
  FileSystem* m_FileSystem;   /* 对全局对象g_FileSystem的引用，该对象负责管理文件系统存储资源 */
  InodeTable* m_InodeTable;   /* 对全局对象g_InodeTable的引用，该对象负责内存Inode表的管理 */
  OpenFileTable* m_OpenFileTable;  /* 对全局对象g_OpenFileTable的引用，该对象负责打开文件表项的管理 */
```

```
enum DirectorySearchMode
{
    OPEN = 0,     /* 以打开文件方式搜索目录 */
    CREATE = 1,   /* 以新建文件方式搜索目录 */
    DELETE = 2    /* 以删除文件方式搜索目录 */
};
```


FileManager
Class
字段
m_FileSystem
m_InodeTable
m_OpenFileTable
rootDirInode
方法
~FileManager
Access
ChDir
ChMod
ChOwn
Close
Creat
FileManager
FStat
Initialize
Link
MakNode
NameI
NextChar
Open
Open1
Owner
Rdwr
Read
Seek
SetCurDir
Stat
Stat1
UnLink
Write
WriteDir
嵌套类型
DirectorySear...
Enum
InodeTable
Class
字段
m_FileSystem
m_Inode
NINODE
方法
~InodeTable
GetFreeInode
IGet
Initialize
InodeTable
IPut
IsLoaded
UpdateInodeTable
IOParameter
Class
字段
m_Base
m_Count
m_Offset
方法
~IOParameter
IOParameter
FileSystem
Class
字段
DATA_ZONE_END_SECTOR
DATA_ZONE_SIZE
DATA_ZONE_START_SECTOR
INODE_NUMBER_PER_SECTOR
INODE_ZONE_SIZE
INODE_ZONE_START_SECTOR
ROOTINO
SUPER_BLOCK_SECTOR_NUMBER
superblk
方法
~FileSystem
Alloc
BadBlock
FileSystem
Free
IAlloc
IFree
Initialize
LoadSuperBlock
Update
SuperBlock
Class
字段
padding
s_flock
s_fmod
s_free
s_fsize
s_ilock
s_inode
s_isize
s_nfree
s_ninode
s_ronly
s_time
方法
~SuperBlock
SuperBlock
Inode
Class
字段
方法
~Inode
Bmap
Clean
CloseI
ICopy
Inode
ITrunc
IUpdate
OpenI
Plock
Prele
ReadI
WriteI
嵌套类型
File
Class
字段
f_count
f_flag
f_inode
f_offset
方法
~File
File
嵌套类型
OpenFileTable
Class
字段
m_File
NFILE
方法
~OpenFileTable
CloseF
FAlloc
OpenFileTable

二、构造模拟二级文件系统

使用一个普通的大文件（**windows或Unix等操作系统所管理的普通文件，称之为一级文件）** **volumeFile来模拟UNIX V6++的一个文件卷**

1. 磁盘文件结构

2. 文件打开结构

3. 文件目录结构

4. 文件操作接口

通过一级文件系统（Windows或Linux）实现volumeFile文件中指定位置的数据读写

主 要 内 容

1 题目

2 设计报告要求

3 提交时间

4 答疑

一、UNIX V6++文件系统代码分析

1. 磁盘文件结构

SuperBlock，DiskInode和FielSystem类的详细说明，包括所有数据成员的说明及FileSystem类中重要成员函数的执行流程。

2. 文件打开结构

Inode，InodeTable、File和OpenFileTable等类的详细说明，包括所有数据成员的说明及重要成员函数的执行流程。

3. 文件目录结构

4. 文件操作接口

FileManager类的详细说明，包括所有数据成员的说明及重要成员函数的执行流程。

一、UNIX V6++文件系统代码分析

5. 实例分析

假如文件Jerry大小为5000字节，且/usr/ast/目录下只有一个Jerry文件，现执行下面的代码：

```
main()
{
    int fd = open("/usr/ast/Jerry", O_RDONLY); //以只读方式打开文件
    char data[500] = "abcde……";
    ……;
    seek(fd, 4500, 0);
    read(fd, data, 500);
    write(fd, data, 500);
    ……;
}
```

（1）详细叙述open系统调用的工作过程？最坏情况下会有几次磁盘操作？最好情况下呢？

一、UNIX V6++文件系统代码分析

5. 实例分析

（2）详细叙述read系统调用的工作过程？最坏情况下会有几次磁盘操作？最好情况下呢？

（3）详细叙述read系统调用的工作过程？最坏情况下会有几次磁盘操作？最好情况下呢？

一、UNIX V6++文件系统代码分析

评分标准

视 1 ~ 4 的完成情况：不及格 ~良

实例分析正确：优

二、构造模拟二级文件系统

1. 磁盘文件结构

说明磁盘文件的结构及存储空间的管理算法，主要算法请示出流程图。

2. 文件打开结构

文件打开结构的主要数据结构说明，主要算法流程。

3. 文件目录结构

4. 文件操作接口

至少包括文件创建、删除、读、写、目录查找等主要算法的说明及流程图。

5. 运行演示

二、构造模拟二级文件系统

6. 实例分析

在你的volumeFile文件模拟的文件系统中，创建文件Jerry，并写入5000个字节，然后执行下面的代码：

```
main()
{
    int fd = open("/usr/ast/Jerry", O_RDONLY);  //以只读方式打开文件
    char data[500] = "abcde……";
    ……;
    seek(fd, 4500, 0);
    read(fd, data, 500);
    write(fd, data, 500);
    ……;
}
```

分别详细叙述open、read和write系统调用的工作过程

二、构造模拟二级文件系统

评分标准

视 1 ~ 5 的完成情况：不及格 ~良

实例分析正确：优

或实现内存高速缓存：优

主 要 内 容

1 题目

2 设计报告要求

3 提交时间

4 答疑

下学期开学后一周内，请班长将所有同学的设计刻成光盘后提交，要求：

（1）每个同学的设计一个文件夹，命名为：

学号_姓名_设计题目

（2）每个文件夹中包括：

题目1：设计报告

题目2：设计报告、源代码、可执行程序

注：不要包含.rar，.zip的任何格式的压缩文件。

所有评优的学生要通过答辩

主 要 内 容

1 题目

2 设计报告要求

3 提交时间

4 答疑

7月21日~8月1日

周一~周五

上午9:00~11:00，下午1:00~3:00

电信学院507