目 录

装 订 线

1 需求分析

1.1 任务概要

实现一个类 UNIX 二级文件系统。

使用一个普通的大文件来模拟 UNIX V6++的一个文件卷。定义自己的磁盘文件结构、文件目录结构、文件打开结构和与之相关的算法。完成内存高速缓存的相关内容。

SuperBlock	Inode 区	文件数据区

1.2 输入形式

输入形式为控制台指令。

文件操作接口对应的输入形式：

指令名	功能	格式	参数	例子
fformat	格式化文件卷	fformat	\	fformat
ls	列目录和文件	ls	\	ls
mkdir	创建目录	mkdir <dirname>	<dirname>：有效目录路径	mkdir mydir mkdir ../mysubdir/mydir
fcreat	新建文件	fcreat <filepath> <mode>	<filepath>：有效文件路径 <mode> ： -r 只读模式 -w 只写模式 -rw -wr 读写模式	fcreat myfile.txt -r fcreat ../subdir/myfile.txt -rw
fopen	打开文件	fopen <filepath> <mode>	<filepath>：有效文件路径 <mode> ： -r 只读模式 -w 只写模式 -rw -wr 读写模式	fopen myfile.txt -r fopen ../subdir/myfile.txt - rw
fclose	关闭文件	fclose <file descriptor>	<file descriptor> ：一个不小于 0 的数字，用来表示一个文件	fclose 1
fread	读文件	fread <file descriptor> [-o <out file name>] <size>	<file descriptor> ：一个不小于 0 的数字，用来表示一个文件，该文件已经被打开	fread 1 100 fread 1 -o myoutfile.txt 100

			<out file name> ：要读出的文件的合法路径 <size> ：要读的字节数	
fwrite	写文件	write <file descriptor> <in file name> <size>	<file descriptor> ：一个不小于 0 的数字，用来表示一个文件，该文件已经被打开 <in file name> ：写入读取的文件路径 <size> ：要写的字节数	fwrite 1 myinfile.txt 100 fwrite 1 mydir/myinfile.txt 10
fseek	定位文件读写指针	fseek <file descriptor> <offset> <origin>	<file descriptor> ：一个不小于 0 的数字，用来表示一个文件，该文件已经被打开 <offset> ：偏移量 <original position> ：0 -从文件头；1 -从当前指针；2 -从文件末尾	fseek 1 100 0
fdelete	删除文件	fdelete <filepath>	<filepath> ：有效文件路径	fdelete myfile.txt -rw fdelete ../subdir/myfile.txt - r
build	系统构建，生成初始化文件结构	build	\	build
exit	退出系统	exit	\	exit
help	查看帮助	help <-cmd>	<-cmd>：要查看的指令名字	help -fdelete

装 订 线

1.3 输出形式

输出形式为控制台输出或输出到文件。

文件操作接口对应的的输出形式：

指令名	情况	输出格式
fformat	格式化系统	\
ls		一行输出一个目录名/文件名
mkdir	成功创建	create directory 创建的目录名 successfully.
	参数数量不足	number of parameters is less than required.Use 'help -mkdir' for more information.
	路径非法	path 路径名 is not a valid path. Use 'help -mkdir' for more information.

	尝试创建失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出
fcreat	成功创建	create file 创建的文件名 successfully.
	参数数量不足	number of parameters is less than required.Use 'help -fcreat' for more information.
	路径非法	path 输入路径名 is not a valid path. Use 'help - fcreat' for more information.
	模式非法	mode '输入模式名' is not a valid mode. "Use 'help - fcreat' for more information.
	尝试创建失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出
fopen	打开成功	open success, return fd=文件标识符
	参数数量不足	number of parameters is less than required.Use 'help - fopen' for more information.
	路径非法	path 输入路径名 is not a valid path. Use 'help - open' for more information.
	模式非法	mode '输入模式名' is not a valid mode. "Use 'help - open' for more information.
	打开失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出
fclose	关闭成功	close file 文件名 successfully.
	fd 不是数字	fd '文件标识符' is not a number. Use 'help - fclose' for more information.
	fd 小于 0	fd should not less than 0.Use 'help -fclose' for more information.
	关闭失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出
fread	读成功	read 读取的字符数 bytes success
	参数数量不足	number of parameters is less than required.Use 'help - fread' for more information.
	fd 不是数字	fd '文件标识符' is not a number. Use 'help - fread' for more information.
	fd 小于 0	fd should not less than 0.Use 'help -fread' for more information.
	size 不是数字	size 输入的字符数 is not a number. Use 'help -fread' for more information"
	size 小于 0	size should not less than 0.Use 'help -fread' for more information.
	尝试读失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出

装 订 线

fwrite	写成功	Write 成功写入的字节数 bytes successfully.
	参数数量不足	number of parameters is less than required.Use 'help -fwrite' for more information.
	fd 不是数字	fd '文件标识符' is not a number. Use 'help -fwrite' for more information.
	fd 小于 0	fd should not less than 0. Use 'help -fwrite' for more information.
	打开外部文件失败	cannot find file '输入文件名'. Use 'help -fwrite' for more information.
	size 不是数字	size 输入的字符数 is not a number. Use 'help -fwrite' for more information"
	size 小于 0	size should not less than 0.Use 'help -fwrite' for more information.
	尝试写失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出
fseek	成功	\
	参数数量不足	number of parameters is less than required.Use 'help - fseek' for more information.
	fd 不是数字	fd '文件标识符' is not a number. Use 'help -fseek' for more information.
	fd 小于 0	fd should not less than 0.Use 'help -fseek' for more information.
	offset 不是数字	offset '偏移量' is not a number. Use 'help -fseek' for more information.
	offset 小于 0	offset should not less than 0.Use 'help -fseek' for more information.
	origin 不是数字	origin '偏移量' is not a number. Use 'help -fseek' for more information.
	origin 不在 0，1，2 之内	origin is in range 0,1,2.Use 'help -fseek' for more information.
	fd 超过最大的打开 fd	error code: 错误码 错误码对应错误信息
fdelete	成功	delete '文件路径名' success.
	参数数量不足	number of parameters is less than required.Use 'help - fdelete' for more information.
	路径非法	path 输入路径名 is not a valid path. Use 'help - open' for more information.
	尝试删除失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出
build	成功	输出每条调用的指令和重要指令的结果
	尝试构建失败	error code: 错误码 错误码对应错误信息 或 严重错误 系统抛出对应错误后退出
exit	/	/
help	/	输出对应提示

装 订 线

1.4 程序功能

模拟 UNIX 二级文件系统，在 CMD 窗口根据用户的输入（详见 1.2），进行二级文件系统文件操作，输出对应的正确错误提示（详见 1.3）。

装 订 线

2 概要设计

2.1 任务分解

2.1.1 输入输出处理

Shell 作为与用户交互的直接接口，对用户的输入进行基本的合法性判断，并输出易懂的用户提示和程序运行结果。

2.1.2 内核运行

内核 Kernel 对 Shell 提供的参数和自身算法结构进行类 UNIX 文件系统的相关操作。其模块和对应的功能如下

- **高速缓存管理 BufferManager**:管理系统中的所有缓存块，包括缓存初始化、分配、释放、读写、刷新。
- **磁盘驱动 DiskDriver**:负责镜像文件 myDisk.img 的直接操作，包括装载、读写。
- **文件管理 FileManager**:提供对文件系统的操作接口，包括文件的打开关闭、读写、读写指针的移动、文件的创建删除。
- **文件系统资源管理 FileSystem**:负责镜像文件的存储空间管理，包括 SuperBlock 的读入更新，Inode 的分配释放，磁盘块的分配释放。
- **打开文件管理 OpenFileManager**:管理系统打开的文件，建立用户与打开文件内核数据的勾连关系。
- **用户 User**:保存、处理 Shell 中内核处理需要的信息，以提供给其他内核模块。

2.2 数据结构定义

以下介绍省略各类类函数、标志位枚举量、常量值

2.2.1 Kernel

```
class Kernel
{
public:
    friend class Shell;
private:
    static Kernel instance;     /* Kernel 单体类实例 */
    BufferManager* m_BufferManager;
```

```
    DiskDriver* m_DiskDriver;
    FileSystem* m_FileSystem;
    FileManager* m_FileManager;
    User* m_User;
};
```

2.2.2 Buf

```
class Buf
{
public:
    unsigned int b_flags;       /* 缓存控制块标志位 */
    Buf* b_forw;
    Buf* b_back;
    int     b_wcount;           /* 需传送的字节数 */
    unsigned char* b_addr;  /*指向该缓存块管理的缓冲区的首地址*/
    int     b_blkno;            /* 磁盘逻辑块号 */
    int     b_error;        /* I/O出错时信息 */
    int     b_resid;          /* I/O出错时尚未传送的剩余字节数 */
};
```

2.2.3 BufferManager

```
class BufferManager
{
private:
    Buf bFreeList;                          /* 缓存队列控制块 */
    Buf m_Buf[NBUF];                        /* 缓存控制块数组 */
    unsigned char Buffer[NBUF][BUFFER_SIZE];    /* 缓冲区数组 */
    std::map<int, Buf*> bMap;               /* 使用的缓存块map */
};
```

2.2.4 DiskDriver

```
class DiskDriver
{
private:
    FILE* fp;    /* 磁盘文件指针 */
};
```

2.2.5 Inode

```
class Inode
{
public:
    unsigned int i_flag; /* 状态的标志位，定义见 enum INodeFlag */
    unsigned int i_mode; /* 文件工作方式信息 */
    int     i_count;     /* 引用计数 */
    int     i_nlink;     /* 文件联结计数，即该文件在目录树中不同路径
名的数量 */
    short   i_dev;       /* 外存 inode 所在存储设备的设备号 */
    int     i_number;    /* 外存 inode 区中的编号 */
    short   i_uid;       /* 文件所有者的用户标识数 */
    short   i_gid;       /* 文件所有者的组标识数 */
    int     i_size;      /* 文件大小，字节为单位 */
    int     i_addr[10];  /* 用于文件逻辑块好和物理块好转换的基本索
引表 */
};
```

2.2.6 DiskInode

```
class DiskInode
{
public:
    unsigned int d_mode; /* 状态的标志位，定义见 enum INodeFlag */
    int     d_nlink;     /* 文件联结计数，即该文件在目录树中不同路
径名的数量 */
    int     d_size;      /* 文件大小，字节为单位 */
    int     d_addr[10];  /* 用于文件逻辑块号和物理块号转换的基本索
引表 */
    short   d_uid;         /* 文件所有者的用户标识数 */
    short   d_gid;         /* 文件所有者的组标识数 */
    int     d_atime;       /* 最后访问时间 */
    int     d_mtime;       /* 最后修改时间 */
};
```

2.2.7 File

```
class File
{
public:
```

```
    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 */
};
```

2.2.8 OpenFiles

```
class OpenFiles
{
private:
    File* ProcessOpenFileTable[NOFILES];
    /* File 对象的指针数组，指向系统打开文件表中的 File 对象 */
};
```

2.2.9 IOParameter

```
class IOParameter
{
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 */
};
```

2.2.10 OpenFileManager

```
class OpenFileTable
{
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。*/
};
```

```
class InodeTable
{
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 */
};
```

2.2.12 FileSystem

```
class SuperBlock
{
public:
    int     s_isize;        /* 外存 Inode 区占用的盘块数 */
    int     s_fsize;        /* 盘块总数 */
    int     s_nfree;        /* 直接管理的空闲盘块数量 */
    int     s_free[NFREE];  /* 直接管理的空闲盘块索引表 */
    int     s_ninode;       /* 直接管理的空闲外存 Inode 数量 */
    int     s_inode[NINODE];  /* 直接管理的空闲外存 Inode 索引表 */
    int s_flock;                // 封锁空闲盘块索引表标志
    int s_ilock;                // 封锁空闲 INode 表标志
    int s_time;                 // 最近一次更新时间
    int     s_fmod;         /* 内存中 super block 副本被修改标志，意味着需要更新外存对应的 Super Block */
    int     s_ronly;        /* 本文件系统只能读出 */
    int     padding[47];    /* 填充使 SuperBlock 块大小等于 1024 字节，占据 2 个扇区 */
};
```

```
class DirectoryEntry
{
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 */
};
```

```
class FileSystem
{
private:
    DiskDriver* m_DiskDriver;
    SuperBlock* m_spb;/*将原来 Mount 类删除，留下一个 SuperBlock*/
    BufferManager* m_BufferManager;
     /* FileSystem 类需要缓存管理模块(BufferManager)提供的接口 */
};
```

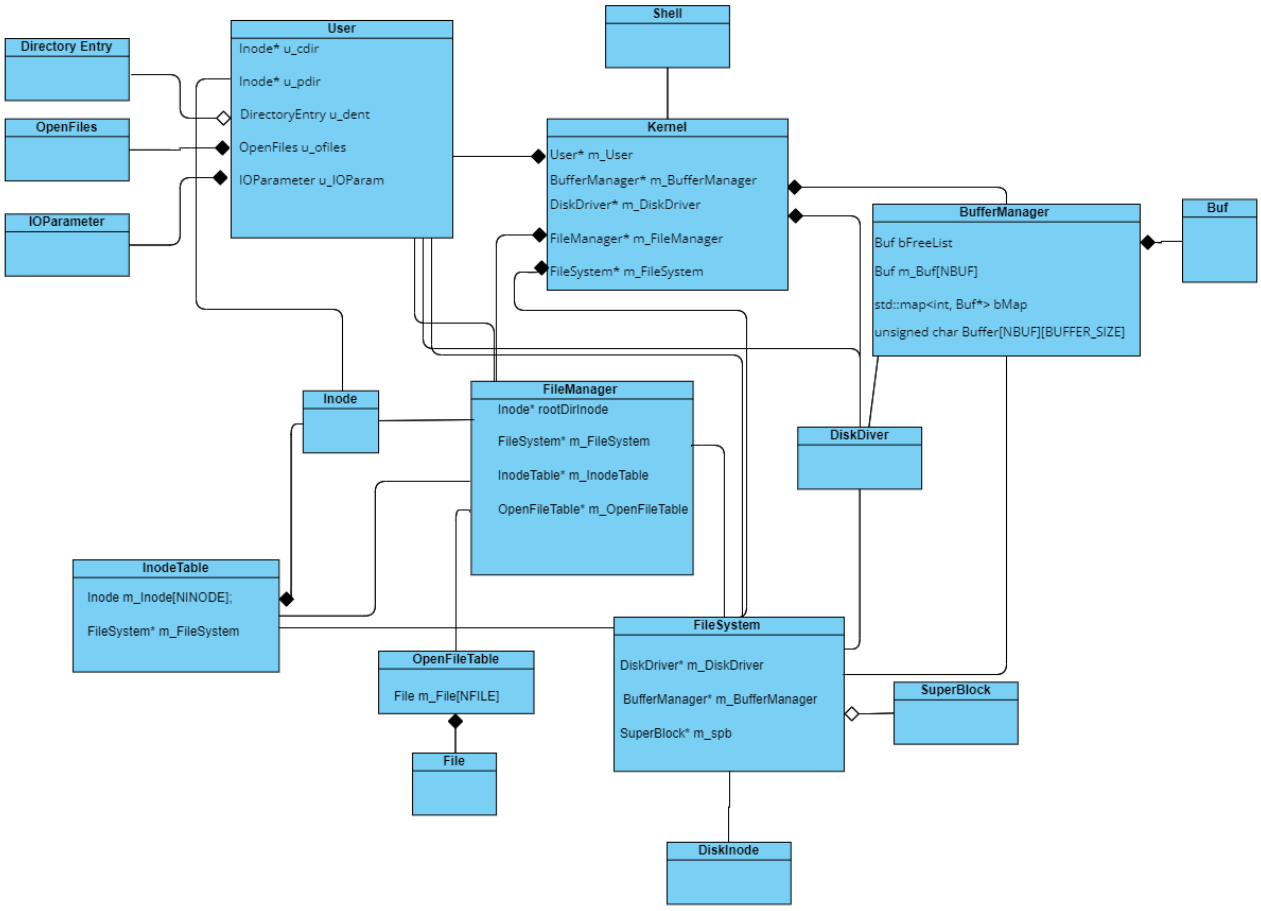
2.2.13 FileManager

```
class FileManager
{
public:
    Inode* rootDirInode; /* 根目录内存 INode */
    FileSystem* m_FileSystem;
    InodeTable* m_InodeTable;
    OpenFileTable* m_OpenFileTable;
};
```

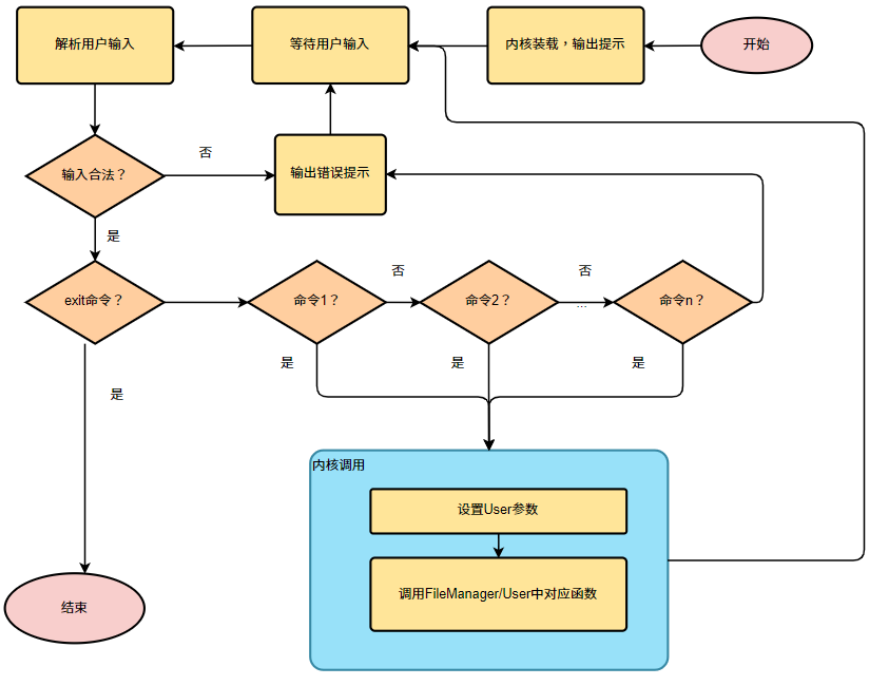
2.2.14 User

```
class User
{
public:
    std::string u_dirp; /* 系统调用参数(一般用于 Pathname)的指针 */
    long u_arg[5];              /* 存放当前系统调用参数 */
    unsigned int    u_ar0[5];       /* 指向核心栈现场保护区中 EAX 寄
存器存放的栈单元，本字段存放该栈单元的地址。*/
    /* 文件系统相关成员 */
    Inode* u_cdir;      /* 指向当前目录的 Inode 指针 */
    Inode* u_pdir;      /* 指向父目录的 Inode 指针 */
    DirectoryEntry u_dent;                  /* 当前目录的目录项 */
    char u_dbuf[DirectoryEntry::DIRSIZ];    /* 当前路径分量 */
    std::string u_curdir;               /* 当前工作目录完整路径 */
    ErrorCode u_error;          /* 存放错误码 */
    OpenFiles u_ofiles;     /* 进程打开文件描述符表对象 */
    IOParameter u_IOParam;  /* 记录当前读、写文件的偏移量，用户目标
区域和剩余字节数参数 */
};
```

2.3 模块间的调用关系


Directory Entry
User
Inode* u_cdir
Inode* u_pdir
DirectoryEntry u_dent
OpenFiles u_ofiles
IOParameter u_IOParam
Shell
OpenFiles
IOParameter
Kernel
User* m_User
BufferManager* m_BufferManager
DiskDriver* m_DiskDriver
FileManager* m_FileManager
FileSystem* m_FileSystem
BufferManager
Buf bFreeList
Buf m_Buf[NBUF]
std::map<int, Buf*> bMap
unsigned char Buffer[NBUF][BUFFER_SIZE]
Buf
Inode
FileManager
Inode* rootDirInode
FileSystem* m_FileSystem
InodeTable* m_InodeTable
OpenFileTable* m_OpenFileTable
DiskDiver
InodeTable
Inode m_Inode[NINODE];
FileSystem* m_FileSystem
FileSystem
DiskDriver* m_DiskDriver
BufferManager* m_BufferManager
SuperBlock* m_spb
SuperBlock
OpenFileTable
File m_File[NFILE]
File
DiskInode

2.4 算法说明


解析用户输入
等待用户输入
内核装载，输出提示
开始
输入合法？
否
输出错误提示
是
exit命令？
命令1？
否
命令2？
否
...
命令n？
是
是
是
是
内核调用
设置User参数
调用FileManager/User中对应函数
结束

总流程图

3 详细设计

分为下面几个部分。

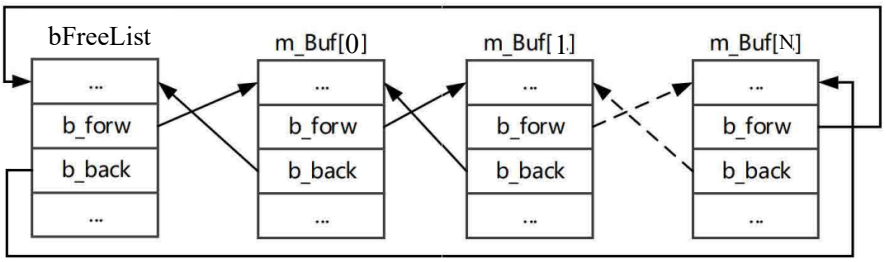
3.1 高速缓存

- BufferManager 重点函数

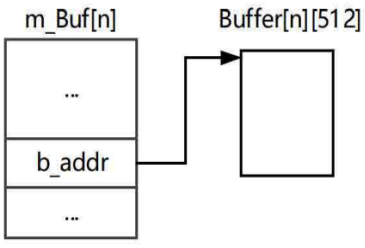
函数名	功能
Initialize	缓存控制块队列的初始化。将缓存控制块中 b_addr 指向相应缓冲区首地址。
GetBlk	申请一块缓存，用于读写设备上的字符块 blkno
Brelse	释放缓存控制块
Bread	读一个磁盘块。blkno 为目标磁盘块逻辑块号
Bwrite	写一个磁盘块
Bdwrite	延迟写磁盘块
ClrBuf	清空缓冲区内容
Bflush	将延迟写的缓存全部输出到磁盘
Format	格式化
NotAvail	从缓存队列中取出第一个缓存块

- **缓存块初始化**

建立双向循环链表，初始化各缓存块的成员变量。



自由缓存队列初始化



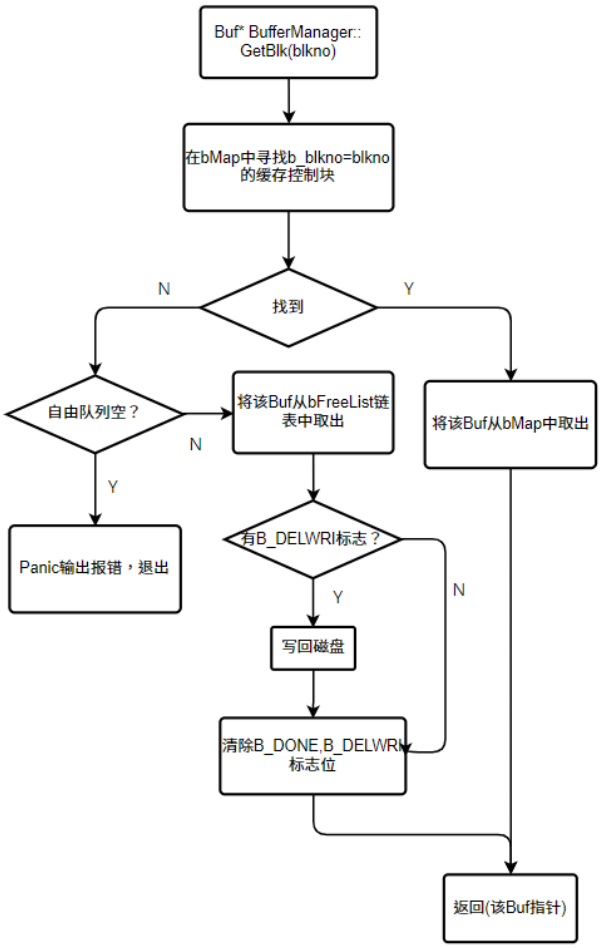
Buf 和缓冲区相勾连

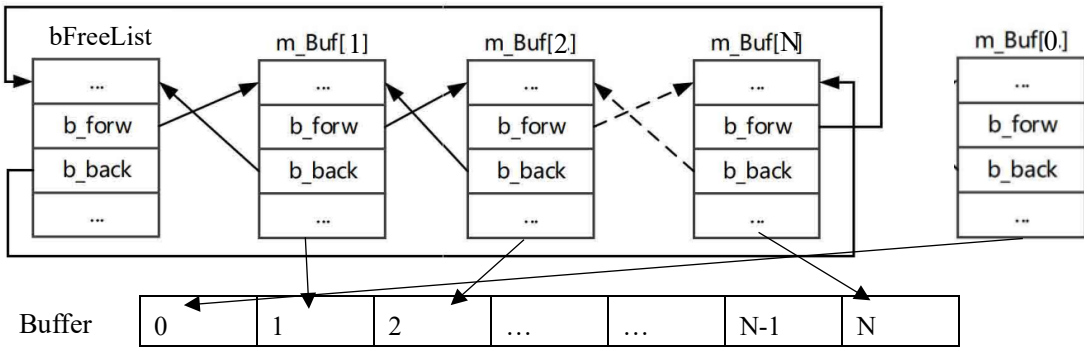
- **缓存块的申请和释放**

缓存块使用 FIFO 算法进行分配释放，std::map<int, Buf*> bMap 记录使用的缓存块写入

的是哪个磁盘块。

申请(GetBlk 函数)的流程图如下





初始化后的队列取出第一个缓存控制块

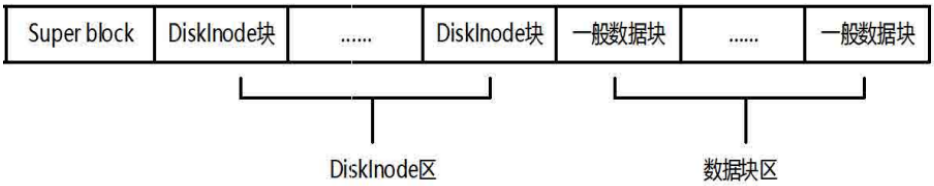
3.2 存储资源管理

FileSystem 重点函数

函数名	功能
Initialize	初始化成员变量
LoadSuperBlock	系统初始化时读入 SuperBlock
Update	将 SuperBlock 对象的内存副本更新到将 SuperBlock 对象

	的内存副本更新到磁盘中去
IAlloc	分配一个空闲外存 INode，一般用于创建新的文件
IFree	释放编号为 number 的外存 INode，一般用于删除文件
Alloc	分配空闲磁盘块
Free	释放编号为 blkno 的磁盘块
Format	格式化

3.2.1 磁盘存储分布



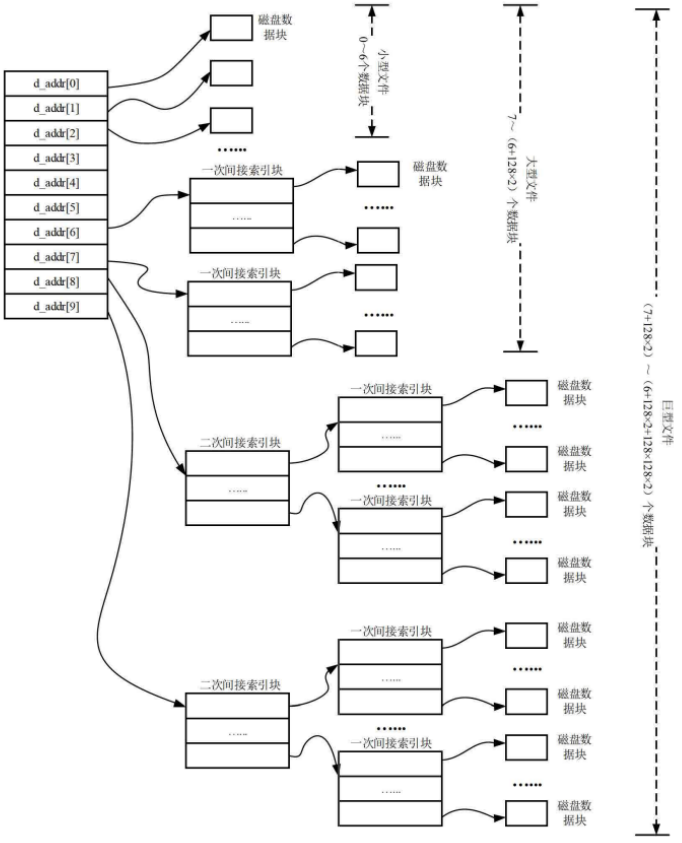
(1) 文件系统超级块(SuperBlock)

(2) 外存索引节点区

(3) 数据区

3.2.2 文件索引结构

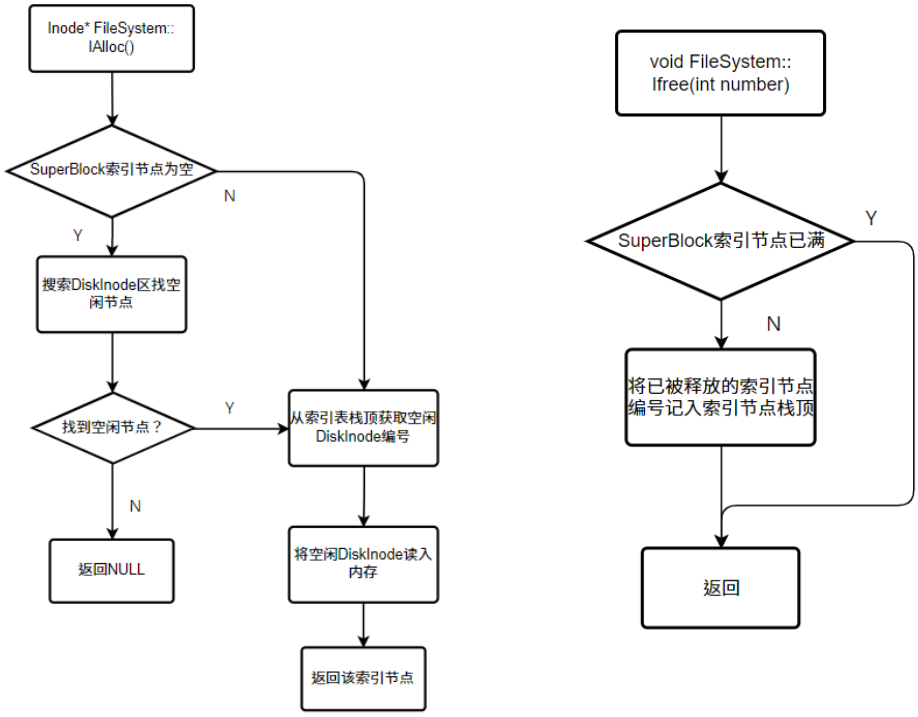
文件保存在数据块上的位置信息包含在外存索引节点中的文件索引表 DiskInode::d_addr[10] 数组中。每一盘块都有唯一的编号，根据文件大小的不同，将文件分为小型、大型、巨型文件。索引结构如下图。



3.2.3 文件系统存储资源管理

SuperBlock 中记录管理外存索引节点和空闲数据块的信息。

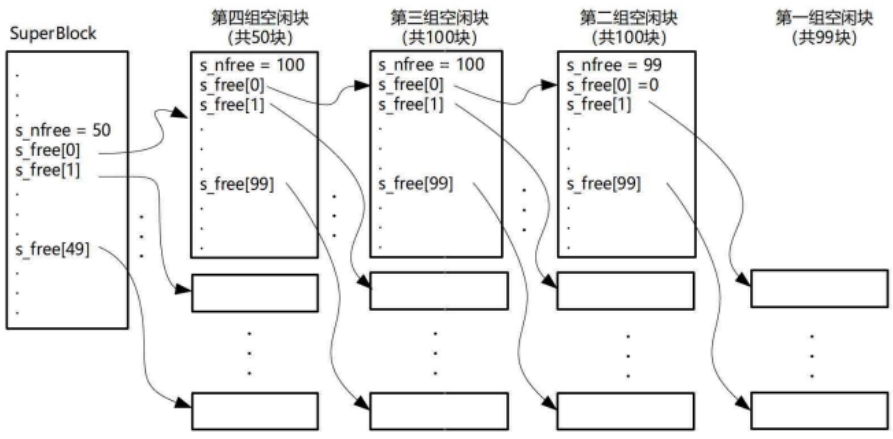
- 空闲外存索引节点的分配和释放



空闲外存索引节点的分配（左）和释放（右）流程图

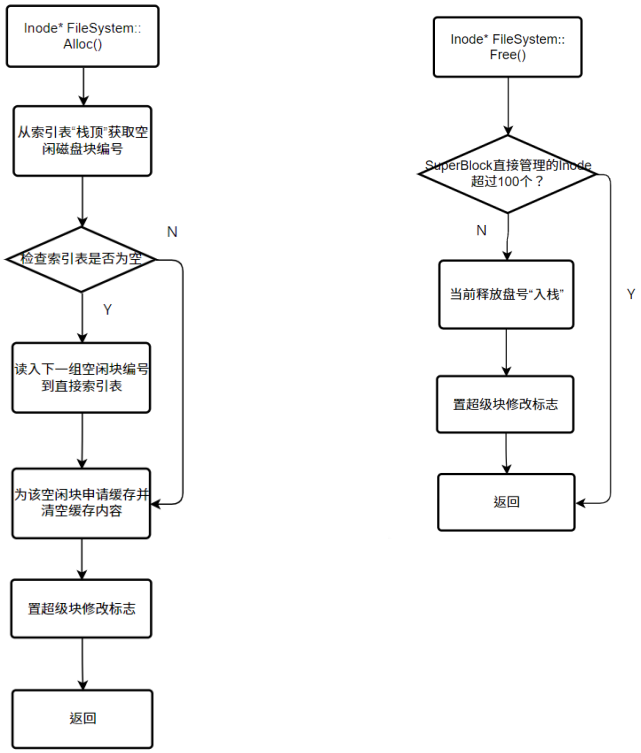
3.2.3 空闲盘块管理

外存中的空闲盘块采用分组链式索引法进行管理。超级块中的空闲块索引表用栈式管理空闲数据块，但是它最多只能直接管理 100 个空闲块。所有空闲块按照每 100 个进行构成一组，最后一组直接由超级块中的空闲索引表进行管理，其余各组的索引表分别存放在它们下一组第一个盘块的开头中。



空闲盘块分组链式索引

- 空闲盘块的分配和释放



空闲盘块的分配（左）和释放（右）流程图

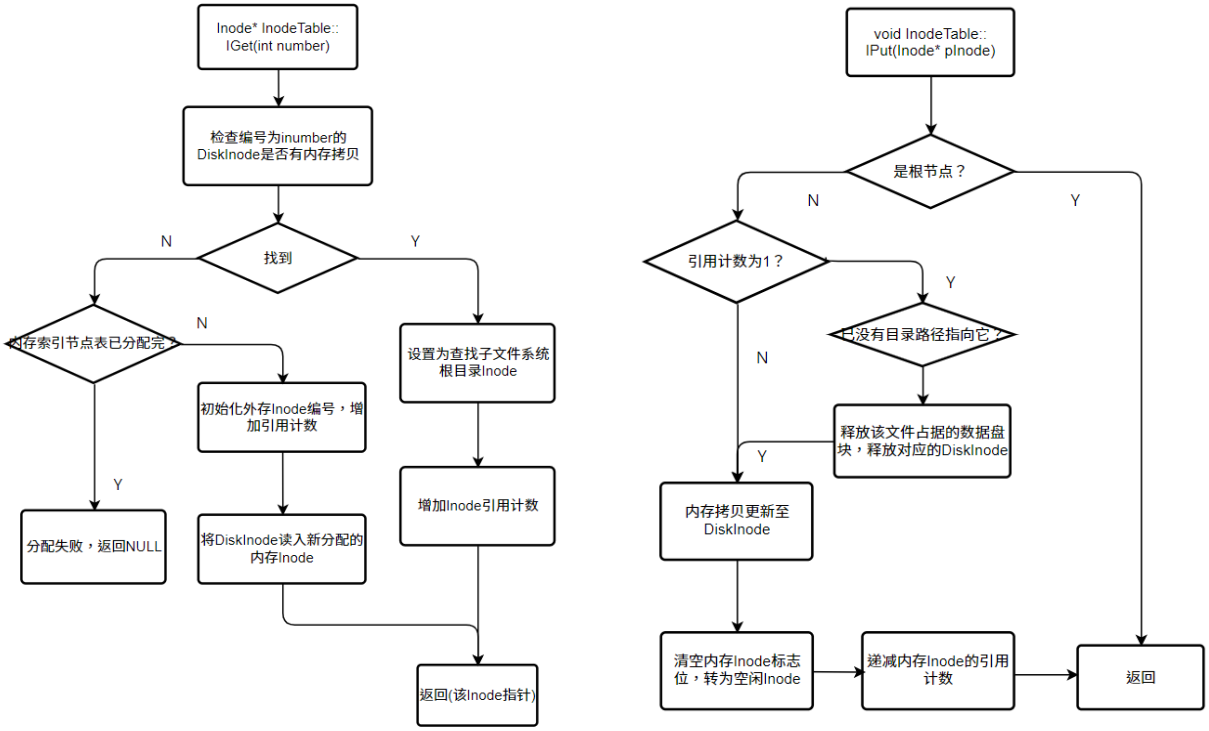
3.3 文件打开结构

内存文件索引节点 Inode，内存文件索引节点表 InodeTable

InodeTable 重点函数

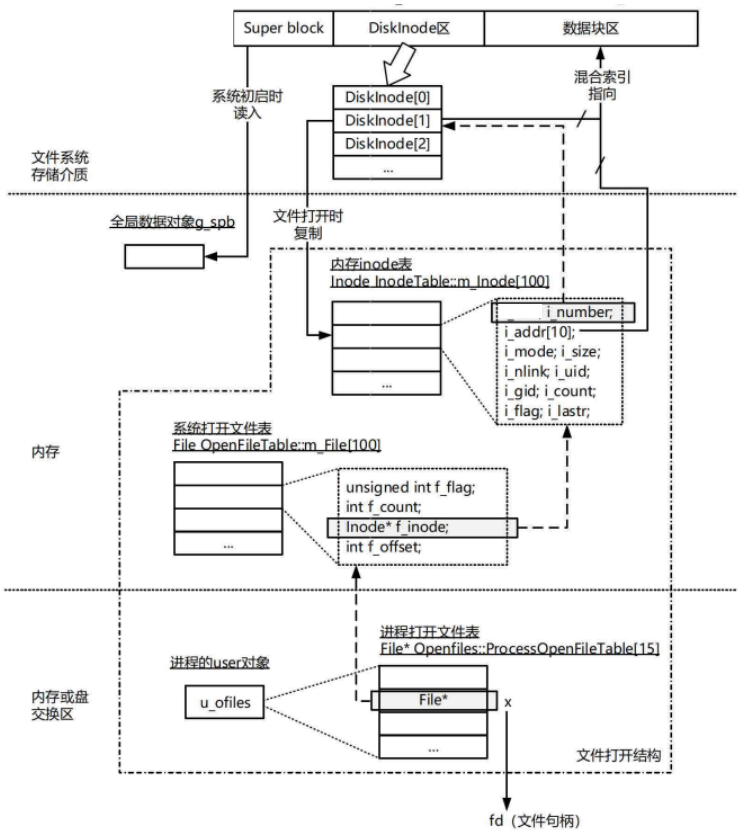
函数名	功能
IGet	根据外存 Inode 编号获取对应 Inode。如果该 Inode 已经在内存中，返回该内存 Inode，如果不在内存中，则将其读入内存后返回该内存 Inode
IPut	减少该内存 Inode 的引用计数，如果此 Inode 已经没有目录项指向它，且无进程引用该 Inode，则释放此文件占用的磁盘块。
UpdateInodeTable	将所有被修改过的内存 Inode 更新到对应外存 Inode 中
IsLoaded	检查设编号为 inumber 的外存 inode 是否有内存拷贝，如果有则返回该内存 Inode 在内存 Inode 表中的索引
GetFreeInode	在内存 Inode 表中寻找一个空闲的内存 Inode
Format	格式化

- 索引节点的获取和释放



索引节点的获取（左）和释放（右）流程图

- 打开文件结构



装 订 线

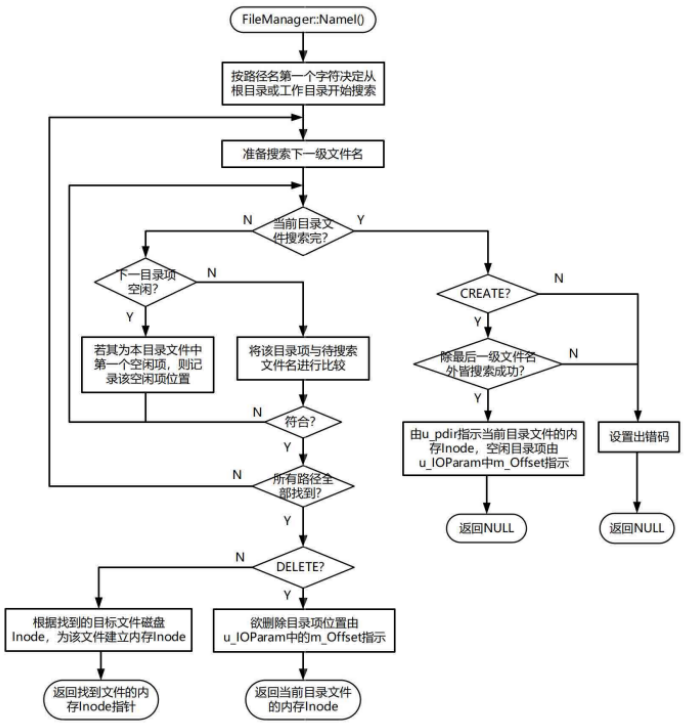
3.4 文件系统的实施

FileManager 重点函数

函数名	功能
Initialize	初始化对全局对象的引用
Open	Open()调用处理过程
Creat	Creat()调用处理过程
Open1	Open()、Creat()系统调用的公共部分
Close	Close()调用处理过程
Seek	Seek()调用处理过程
Read	Read()调用处理过程
Write	Write()调用处理过程
Rdwr	读写调用公共部分代码
NameI	目录搜索，将路径转化为相应的 Inode，返回 Inode
MakNode	被 Creat()调用使用，用于为创建新文件分配内核资源
WriteDir	向父目录的目录文件写入一个目录项
Access	检查对文件或目录的搜索、访问权限，作为调用的辅助函数
UnLink	删除文件
ChDir	改变当前工作目录
Format	格式化

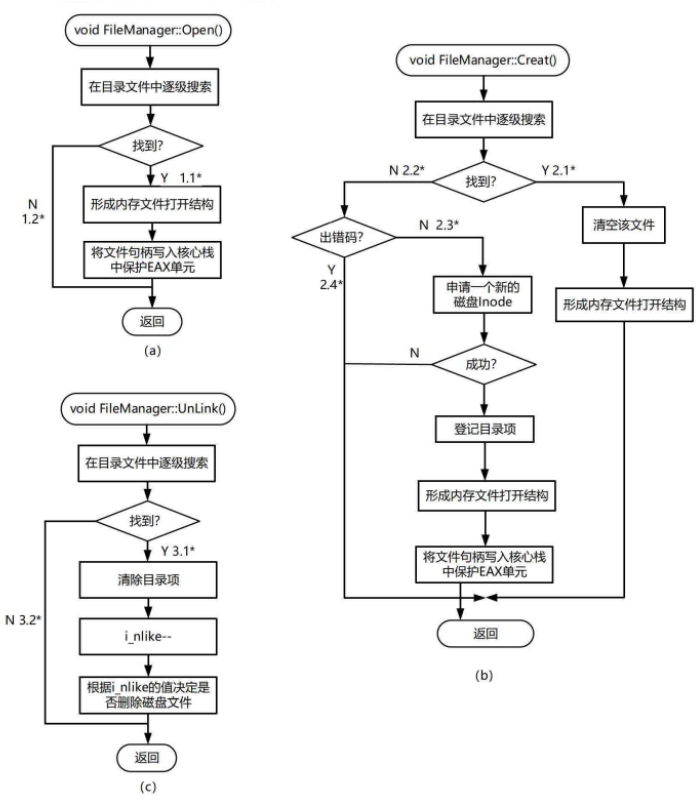
- 目录搜索

目录结构 DirectoryEntry 类，采用树形目录结构



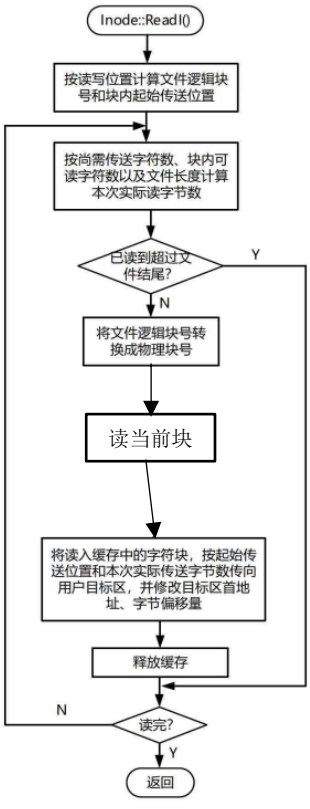
目录搜索(NameI)流程图

- 文件的打开、创建、撤销

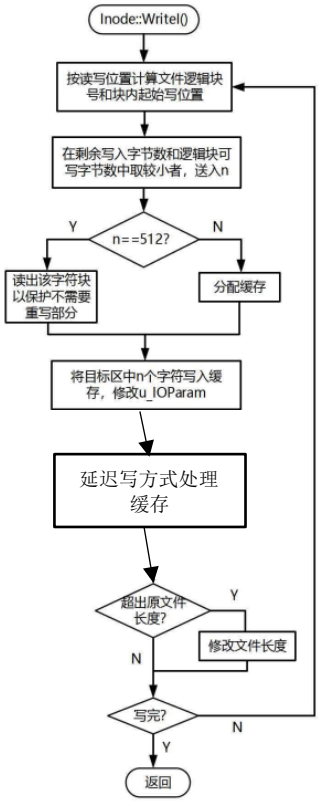


文件的打开、创建、撤销流程图

- 文件的读写



INode::ReadI()函数流程

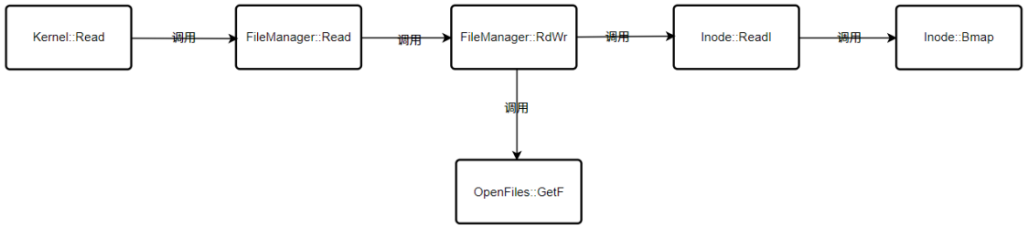


INode::WriteI()函数流程

Inode 重点函数

函数名	功能
ReadI	根据 Inode 对象中的物理磁盘块索引表，读取相应文件数据
WriteI	根据 Inode 对象中的物理磁盘块索引表，将数据写入文件
Bmap	将文件的逻辑块号转换成对应的物理盘块号
IUpdate	更新外存 Inode 的最后的访问时间、修改时间
ITrunc	释放 Inode 对应文件占用的磁盘块
Clean	清空 Inode 对象中的数据
ICopy	将包含外存 Inode 字符块中信息拷贝到内存 Inode 中

读文件的重点函数调用关系如下，写文件与之类似。

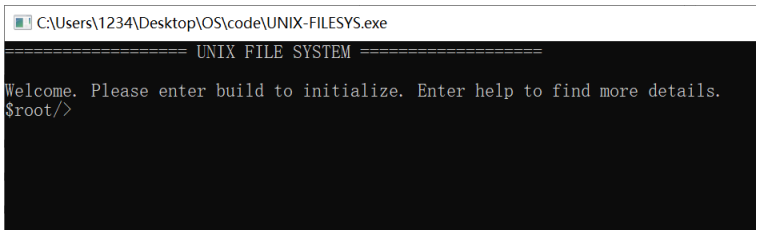


4 运行结果分析

4.1 实验环境

宿主机：Windows10 64 位
编译器：gcc version 9.2.0 (MinGW.org GCC Build-2)

4.2 运行结果及分析



进入 exe 后的界面

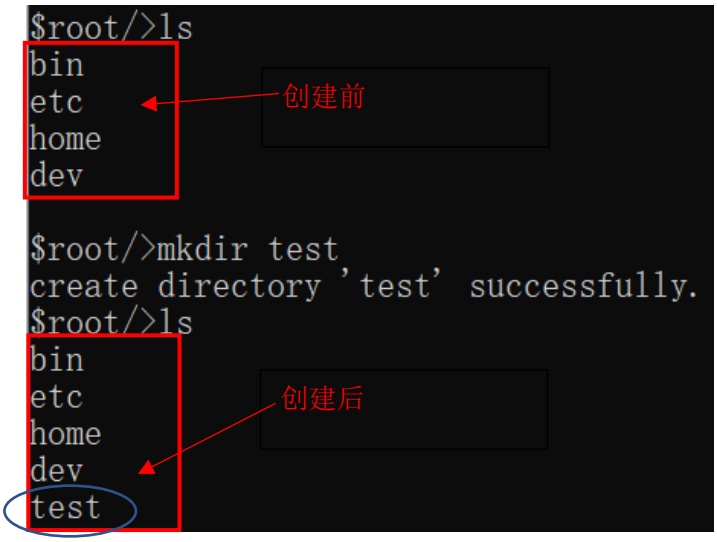
- 自动构建

  输入 build 指令，build 指令是一组指令的集合。 build 指令开始时输出”build start”指示，若构建出错输出”build error”及对应错误提示，build 成功输出“build success”

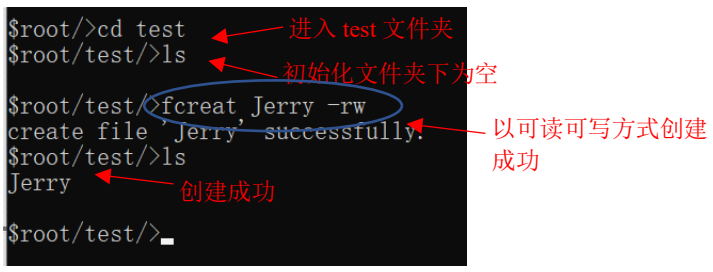
```
Welcome. Please enter build to initialize. Enter help to find more details.
$root/>build
build start
$root/>fformat
$root/>mkdir bin
$root/>mkdir etc
$root/>mkdir home
$root/>mkdir dev
$root/>ls
bin
etc
home
dev
$root/>cd home
$root/home/>mkdir texts
$root/home/>mkdir reports
$root/home/>mkdir photos
$root/home/>ls
texts
reports
photos
$root/home/>cd texts
$root/home/texts/>fcreat ReadMe.txt -rw
$root/home/texts/>fopen ReadMe.txt -rw
open success, return fd=8
$root/home/texts/>fseek 8 0 0
$root/home/texts/>fwrite 8 ../ReadMe.txt 8000
write 8000 bytes successfully.
$root/home/texts/>fclose 8
$root/home/texts/>cd ..
$root/home/>cd reports
$root/home/reports/>fcreat report.pdf -rw
$root/home/reports/>fopen report.pdf -rw
open success, return fd=9
$root/home/reports/>fseek 9 0 0
$root/home/reports/>fwrite 9 ../report.pdf 8000
write 8000 bytes successfully.
$root/home/reports/>fclose 9
$root/home/reports/>cd ..
$root/home/>cd photos
$root/home/photos/>fcreat star.png -rw
$root/home/photos/>fopen star.png -rw
open success, return fd=10
$root/home/photos/>fseek 10 0 0
$root/home/photos/>fwrite 10 ../star.png 4000
write 4000 bytes successfully.
$root/home/photos/>fclose 9
$root/home/photos/>cd /
build success.
```

成功 build 结果

- 在根目录下新建文件/test/Jerry

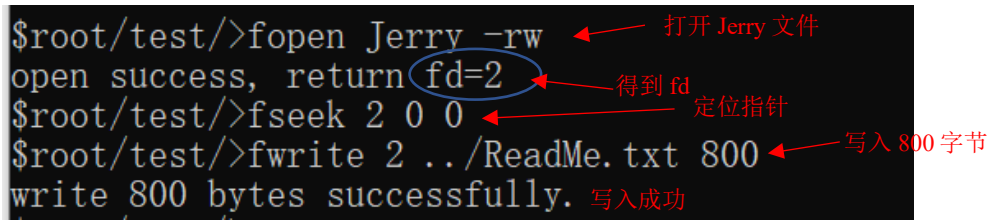


- 创建 test 文件夹过程



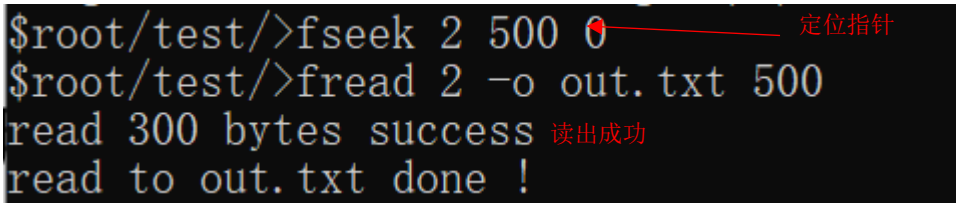
创建 Jerry 文件过程

- 打开/test/Jerry，写入 800 个字节

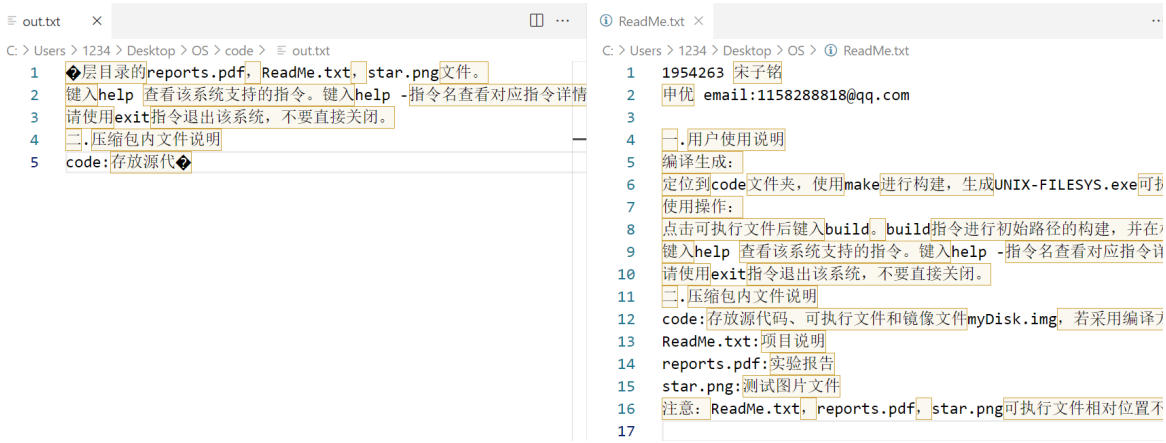


- 将文件读写指针定位到第 500 字节，读出 500 个字节写入 out.txt

  因读到 300 个字节时已达文件末尾，所以成功读取的字节数是 300



cmd 指令

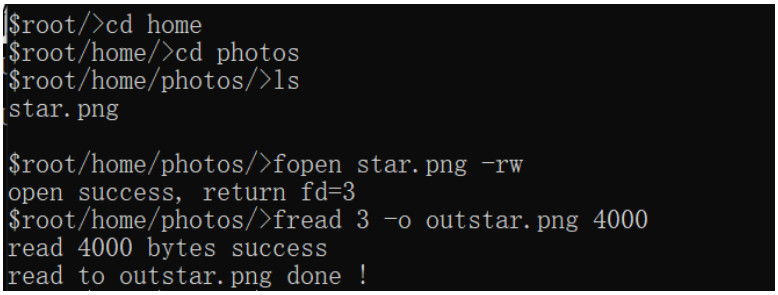


读出（左）写入（右）结果比对

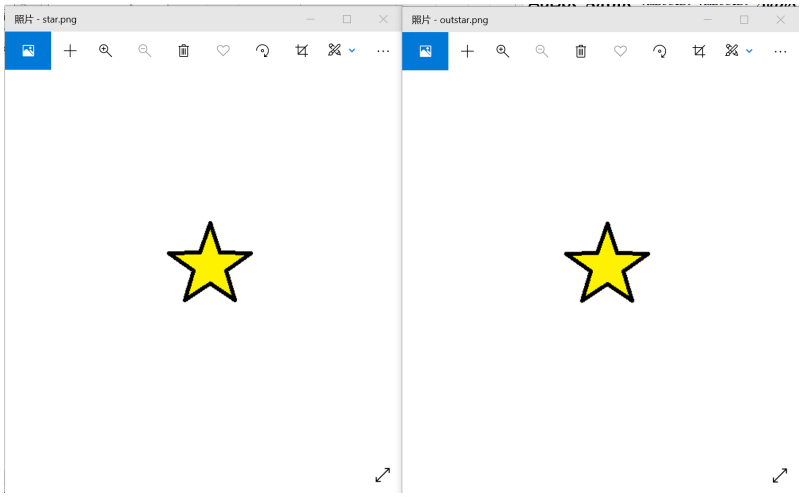
读出的乱码是因为一个汉字占 2 个字节，而恰好只读了一个字节，无法成功编码

- 读写图片

  读出在 build 时已写入的图片



cmd 指令



原图（左）与结果（右）比对

- 部分内核错误处理

  不存在的路径

```
$root/home/photos/>cd notdir
error code: 20
 Not a directory
```

  权限错误（如:读写模式打开只读文件）

```
$root/test/>fcreat onlyread.txt -r
create file 'onlyread.txt' successfully.
$root/test/>fopen onlyread.txt -rw
error code: 13
 Permission denied
```

  删除不存在的文件

```
$root/test/>fdelete notfile
error code: 2
 No such file or directory
```

5 用户使用说明

5.1 编译生成

定位到 code 文件夹，使用 make 进行构建，生成 UNIX-FILESYS.exe 可执行文件

```
C:\Users\1234>cd C:\Users\1234\Desktop\OS\code

C:\Users\1234\Desktop\OS\code>mingw32-make
g++ -std=c++11 -w  -c -o obj/main.o main.cpp
g++ -std=c++11 -w  -c -o obj/BufferManager.o BufferManager.cpp
g++ -std=c++11 -w  -c -o obj/DiskDriver.o DiskDriver.cpp
g++ -std=c++11 -w  -c -o obj/File.o File.cpp
g++ -std=c++11 -w  -c -o obj/FileManager.o FileManager.cpp
g++ -std=c++11 -w  -c -o obj/FileSystem.o FileSystem.cpp
g++ -std=c++11 -w  -c -o obj/INode.o INode.cpp
g++ -std=c++11 -w  -c -o obj/Kernel.o Kernel.cpp
g++ -std=c++11 -w  -c -o obj/OpenFileManager.o OpenFileManager.cpp
g++ -std=c++11 -w  -c -o obj/Shell.o Shell.cpp
g++ -std=c++11 -w  -c -o obj/User.o User.cpp
g++ -std=c++11 -w  -c -o obj/Utility.o Utility.cpp
g++ -o UNIX-FILESYS obj/main.o obj/BufferManager.o obj/DiskDriver.o obj/File.o obj/FileManager.o obj/FileSystem.o obj/IN
ode.o obj/Kernel.o obj/OpenFileManager.o obj/Shell.o obj/User.o obj/Utility.o
```

  生成的.o 文件放在 obj 文件夹中

生成的可执行文件 UNIX-FILESYS 在 code 文件夹下

也可省略编译步骤，直接使用压缩包内的可执行文件，在 Windows 系统下双击该可执行文件即可运行。

5.2 使用操作

点击可执行文件后键入 build。build 指令进行初始路径的构建，并在相应路径创建读入上层目录的 reports.pdf，ReadMe.txt，star.png 文件。结果见 4.2

键入 help 查看该系统支持的指令。

```
$root/>help
Unix file system support commands:
fformat
exit
cd
ls
mkdir
fcreat
fdelete
fopen
fclose
fseek
fread
fwrite
Type `help -name' to find out more about the command `name'.
```

键入 help -指令名查看对应指令详情

```
fread       :
discription :  read a opened file
usage       :  fread <file descriptor> [-o <out file name>] <size>
parameter   :  <file descriptor> a valid number to describle a file
               <out file name> a valid file path to output, default option is shell.
               <size> read bytes
example     :  fread 1 100
            :  fread 1 -o myoutfile.txt 100
```

输入相应指令即可进行操作。

请使用 exit 指令退出该系统，不要直接关闭。

参考资料

[1] UNIX V6++ 源代码

[2] 同济大学操作系统原理讲义

[3] BaiJiazm/FileSystem: 类 Unix 文件系统设计与实现 二级文件系统 (github.com)

[4] Operating-System-Course/code at main · teamwong111/Operating-System-Course (github.com)