



同濟大學
TONGJI UNIVERSITY

同济大学操作系统课程设计 UNIX文件系统实验报告

姓 名 郑博远

学 号 2154312

学 院 电子与信息工程学院

专 业 计算机科学与技术

授课老师 方 钰

日 期 2024 年 5 月

目 录

1.	实验概述.....	5
1.1.	实验概述.....	5
1.2.	实验要求.....	5
1.3.	实验环境.....	7
2.	需求分析.....	7
2.1.	输入输出形式.....	7
2.1.1.	程序输入.....	7
2.1.2.	程序输出.....	8
2.2.	程序功能.....	8
3.	概要设计.....	9
3.1.	任务分解.....	9
3.1.1.	内核模块：OSKernel.....	9
3.1.2.	一级文件读写模块：BlockDevice、DeviceManager.....	10
3.1.3.	高速缓存模块：Buf、BufferManager.....	10
3.1.4.	文件管理模块：FileSystem、Inode、FileManager、File、OpenFileManager.....	10
3.1.5.	用户模块：User.....	10
3.1.6.	交互模块：Shell、SystemCall.....	11
3.1.7.	日志模块：Logger.....	11
3.2.	数据结构定义.....	11
3.2.1.	一级（虚拟磁盘）文件存储结构.....	11
3.2.2.	SuperBlock 超级块结构.....	13
3.2.3.	DiskInode 结构.....	14
3.2.4.	DirectoryEntry 目录结构.....	16
3.2.5.	打开文件结构.....	17
3.2.6.	高速缓存块结构.....	19

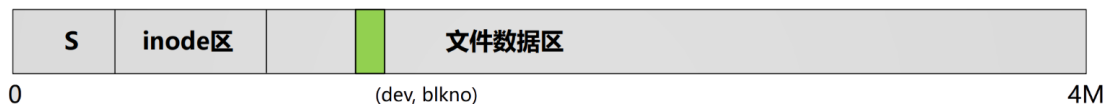
3.3.	模块间的调用关系	22
3.4.	算法说明	22
3.4.1.	高速缓存管理算法	22
3.4.2.	Inode 节点分配回收算法	23
3.4.3.	文件数据盘块分配回收算法	23
3.4.4.	目录项搜索算法	24
3.4.5.	数据库索引映射算法	24
4.	详细设计	24
4.1.	重点函数与重点变量	24
4.1.1.	VirtualFileDevice	24
4.1.2.	BufferManager	25
4.1.3.	Buf.....	26
4.1.4.	FileManager	26
4.1.5.	OpenFileTable	27
4.1.6.	InodeTable	27
4.1.7.	Inode.....	28
4.1.8.	FileSystem.....	29
4.1.9.	SuperBlock.....	30
4.1.10.	User	30
4.1.11.	OSKernel.....	31
4.1.12.	SystemCall	31
4.2.	重点功能程序流程图	32
4.2.1.	高速缓存块管理	32
4.2.2.	Inode 节点分配回收	34
4.2.3.	文件数据盘块分配回收	36
4.2.4.	目录项搜索	37

4.2.5.	数据块索引映射	38
4.2.6.	文件打开与关闭	38
4.2.7.	文件读写流程	39
5.	运行结果分析	40
5.1.	程序运行结果展示	40
5.1.1.	格式化文件卷	40
5.1.2.	用 mkdir 命令创建目录	42
5.1.3.	存储文件	42
5.2.	测试给定命令及输出	44
5.3.	测试其他命令及输出	48
5.3.1.	help	48
5.3.2.	ls	48
5.3.3.	cd	49
5.3.4.	cp	49
5.3.5.	mv	50
5.3.6.	fopen	50
5.3.7.	fdelete	51
5.3.8.	fread、fwrite	51
5.3.9.	tree	51
5.3.10.	fformat	52
5.3.11.	quit	52
6.	用户使用说明	52
6.1.	使用界面	52
6.2.	注意事项	53
7.	参考文献	54

1. 实验概述

1.1. 实验概述

使用一个普通的大文件（如 `c:\myDisk.img`，称之为一级文件）来模拟 UNIX V6++ 的一个文件卷（把一个大文件当一张磁盘用）。磁盘存储的信息以块为单位。每块 512 字节。



1.2. 实验要求

1. 磁盘文件结构:

- 定义自己的磁盘文件结构
- SuperBlock 结构
- 磁盘 Inode 节点结构，包括：索引结构
- 磁盘 Inode 节点的分配与回收算法设计与实现
- 文件数据区的分配与回收算法设计与实现

2. 在该逻辑磁盘上定义二级文件系统结构:

- SuperBlock 及 Inode 区所在位置及大小
- Inode 节点
- 数据结构定义：注意大小，一个盘块包含整数个 Inode 节点
- Inode 区的组织（给定一个 Inode 节点号，怎样快速定位）

- 索引结构：多级索引结构的构成，索引结构的生成与检索过程

3. 文件系统的目录结构：

- 目录文件的结构
- 目录检索算法的设计与实现
- 目录结构增、删、改的设计与实现

4. 文件打开结构：

- 文件打开结构的设计：内存 Inode 节点，File 结构？进程打开文件表？
- 内存 Inode 节点的分配与回收
- 文件打开过程
- 文件关闭过程

5. 文件操作接口：

- fformat: 格式化文件卷
- ls: 列目录
- mkdir: 创建目录
- fcreat: 新建文件
- fopen: 打开文件
- fclose: 关闭文件
- fread: 读文件
- fwrite: 写文件
- flseek: 定位文件读写指针
- fdelete: 删除文件
- 其他程序接口

1.3. 实验环境

硬件配置：联想小新 14Pro ACH 2021

系统环境：Windows11

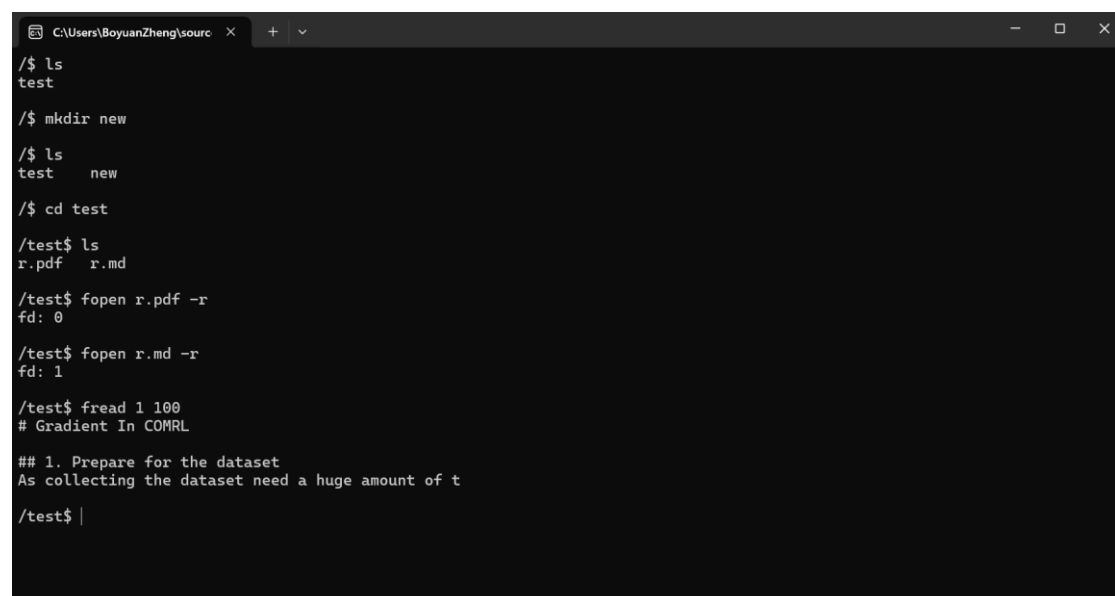
开发环境：Visual Studio 2022

2. 需求分析

2.1. 输入输出形式

2.1.1. 程序输入

本程序通过命令行进行输入输出交互。程序输入方面，支持 `mkdir`、`ls`、`fdelete`、`fopen`、`fcreat`、`fread`、`fwrite`、`cp` 等指令（详见后文），通过控制台输入进行交互。程序交互界面示意图如下：



```
C:\Users\BoyuanZheng\source > /$ ls
test

C:\Users\BoyuanZheng\source > /$ mkdir new

C:\Users\BoyuanZheng\source > /$ ls
test    new

C:\Users\BoyuanZheng\source > /$ cd test

C:\Users\BoyuanZheng\source\test > /test$ ls
r.pdf   r.md

C:\Users\BoyuanZheng\source\test > /test$ fopen r.pdf -r
fd: 0

C:\Users\BoyuanZheng\source\test > /test$ fopen r.md -r
fd: 1

C:\Users\BoyuanZheng\source\test > /test$ fread 1 100
# Gradient In COMRL

## 1. Prepare for the dataset
As collecting the dataset need a huge amount of t

C:\Users\BoyuanZheng\source\test > /test$ |
```

2.1.2. 程序输出

一方面，程序通过如上图所示的命令行进行输出。输出内容包含 `fread` 读到的文件内容、`fopen` 提供的句柄信息等。此外，也会显示错误提示信息（如权限不够、不是目录等）以及命令的使用提示信息。

另一方面，`DeviceManager`、`BufferManager` 等类也会通过 `Logger` 类维护日志，记录下指令执行的关键信息以及方便调试的错误信息等。程序记录的日志信息被填写在 `logfile.txt` 中。具体格式如下图：

```

2024-03-27 19:15:52[INFO] Executing Bread, dev: 0, blkno: 81919
2024-03-27 19:15:52[INFO] Use free buffer 11.
2024-03-27 19:15:52[INFO] Get buffer 11 from BFreeList
2024-03-27 19:15:52[INFO] Read Virtual Disk, blkno:81919 addr:0x27ffe00 count:512
2024-03-27 19:15:52[INFO] Brelse buffer 11, blkno:81919
2024-03-27 19:15:52[INFO] Executing Bread, dev: 0, blkno: 2
2024-03-27 19:15:52[INFO] Found block 2 at buffer 12. Reusing...
2024-03-27 19:15:52[INFO] Get buffer 12 from BFreeList
2024-03-27 19:15:52[INFO] Brelse buffer 12, blkno:2
2024-03-27 19:15:52[INFO] Mapping lbn:0 into blkno:81919
2024-03-27 19:15:52[INFO] Executing Bread, dev: 0, blkno: 81919
2024-03-27 19:15:52[INFO] Found block 81919 at buffer 11. Reusing...
2024-03-27 19:15:52[INFO] Get buffer 11 from BFreeList
2024-03-27 19:15:52[INFO] Executing delayed write at buffer 11
2024-03-27 19:15:52[INFO] Brelse buffer 11, blkno:81919
2024-03-27 19:15:52[INFO] Use free buffer 10.
2024-03-27 19:15:52[INFO] Get buffer 10 from BFreeList
2024-03-27 19:15:52[INFO] Executing delayed write at buffer 10
2024-03-27 19:15:52[INFO] Brelse buffer 10, blkno:81918
2024-03-27 19:15:52[INFO] Mapping lbn:0 into blkno:81918
2024-03-27 19:15:52[INFO] Found block 81918 at buffer 10. Reusing...
2024-03-27 19:15:52[INFO] Get buffer 10 from BFreeList
2024-03-27 19:15:52[INFO] Write Virtual Disk, blkno:81918 addr:0x27ffc00
2024-03-27 19:15:52[INFO] Brelse buffer 10, blkno:81918
2024-03-27 19:15:52[INFO] Use free buffer 9.
2024-03-27 19:15:52[INFO] Get buffer 9 from BFreeList
2024-03-27 19:15:52[INFO] Executing delayed write at buffer 9
2024-03-27 19:15:52[INFO] Brelse buffer 9, blkno:81917
2024-03-27 19:15:52[INFO] Mapping lbn:1 into blkno:81917
2024-03-27 19:15:52[INFO] Found block 81917 at buffer 9. Reusing...
2024-03-27 19:15:52[INFO] Get buffer 9 from BFreeList
    
```

2.2. 程序功能

程序支持 `mkdir`、`ls`、`fdelete`、`fopen`、`fcreat`、等多条命令，具体解释如下（浅灰底色为课设要求之外额外实现的命令）：

命令	命令格式	说明
<code>help</code>	<code>help</code>	展示所有指令使用说明

fformat	fformat	格式化系统
ls	ls -<mode>	展示某目录下的所有文件（夹）
mkdir	mkdir <path>	创建文件夹
fcreat	fcreat <path>	创建文件
fopen	fopen <path> -<mode>	打开文件（-r 、-w 表示读/写模式打开）
fclose	fclose <fd>	关闭文件句柄
fread	fread <fd> <count> [>> <str>]	读取句柄为 fd 的文件 count 字节
fwrite	fwrite <fd> <str> fwrite <fd> << <str>	向句柄为 fd 的文件写入字符串 str
lseek	lseek <fd> <offset> -<mode>	调整句柄 fd 的文件指针（-b、-c、-e 三种模式分别对应文件头、当前、文件尾）
fdelete	fdelete <path>	删除文件
cd	cd <dir>	切换到目标文件夹下
cp	cp <path1> <path2>	复制文件（支持文件系统内外复制，路径前加\$表示系统外部路径）
mv	mv <path1> <path2>	移动文件
tree	tree <path>	展示目录树
quit	quit	退出系统

3. 概要设计

3.1. 任务分解

3.1.1. 内核模块：OSKernel

在本程序中，所有操作都在核心态下进行。OSKernel 类用于封装所有内核相关的全局类实例对象，包括 BufferManager、DeviceManager、FileManager、FileSystem、User 等。OSKernel 类在内存中为单体模式，保证内核中封装各内核模块的对象都只有一个副本。

3.1.2. 一级文件读写模块: BlockDevice、DeviceManager

BlockDevice 表示块设备基类。在本次实验中, 重新设计了继承块设备基类的一级文件虚拟设备派生类 VirtualFileDevice, 通过重载 Strategy 方法进行对一级文件的读写。由于只有一级文件这一虚拟块设备, DeviceManger 仅需要负责对其构造及析构即可。

3.1.3. 高速缓存模块: Buf、BufferManager

Buf 类是对高速缓存块结构以及各个标记的定义, BufferManager 是对高速缓存管理的实现, 包括缓存块的分配、更新、读写等操作。

3.1.4. 文件管理模块: FileSystem、Inode、FileManager、File、OpenFileManager

Filesystem 定义了一级(虚拟磁盘)文件中的存储结构, 包括对超级块的定义, 以及超级块、Inode 区、文件区的划分, 此外还负责对文件系统进行初始化以及对 Inode 和磁盘块的分配与释放。Inode 定义了内存索引节点结构以及其存储在磁盘中 DiskInode 结构。FileManager 模块中则为 Read、Write 等系统调用实现了底层的具体处理过程, 包括 Access、ChDir、MakNode、NameI 等与 Inode 以及文件有关的具体操作。

OpenFileManager 负责打开文件结构的定义与管理。其中打开文件 File 对象, 打开文件表 OpenFileTable, 内存 InodeTable 表。该部分实现了文件打开过程相关结构和管理过程。

3.1.5. 用户模块: User

User 类定义了与用户相关的信息及行为。由于多用户不是本次课程设计的重点, 其主要用于记录当前目录信息, 以及为(伪)系统调用提供 u_arg 的参数传递和 u_ar0 的返回值传递途径。

3.1.6. 交互模块：Shell、SystemCall

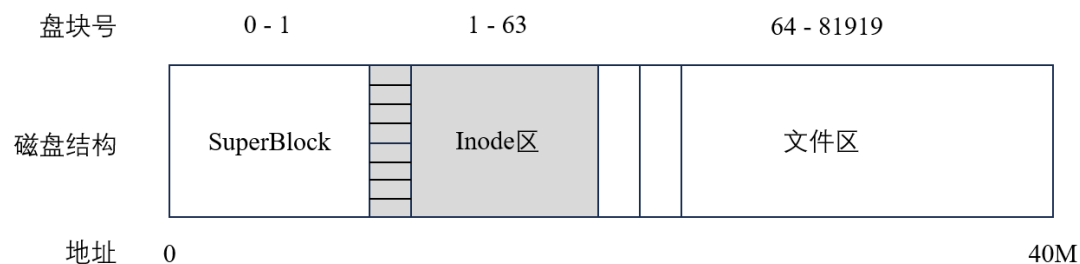
Shell 模块中，每条指令通过继承虚基类 `Command` 进行具体的实现。在 Unix V6++ 中，通常是通过钩子函数进行传递参数和系统调用的发起，再通过对应的 `SystemCall` 函数调用底层函数进行处理。此处的（伪）`SystemCall` 将两件事情合二为一，由 `Command` 类进行调用，负责装填 `u_arg` 信息并调用对应底层模块，并将 `u_ar0` 中的信息进行返回处理。

3.1.7. 日志模块：Logger

`Logger` 类通过单例模式实现。各个模块都可以调用 `info`、`warning`、`error` 三个静态成员函数进行日志输出。

3.2. 数据结构定义

3.2.1. 一级（虚拟磁盘）文件存储结构



一级（虚拟磁盘）文件共 40MB。磁盘中存储的信息以块为单位，每块 512 字节。其被划分为三个部分，具体包括 `SuperBlock` 超级块、`Inode` 区以及文件区，各部分大小与起止盘块号记录在 `FileSystem` 中。

`SuperBlock` 区占用 0、1 两个盘块，共 1024 字节。其主要用于记录磁盘中空闲的资源，包括空闲 `Inode` 以及文件区空闲的盘块。

Inode 区占用 1-63 号盘块，其中每个盘块存储 8 个 DiskINode 对象，每个 DiskINode 占 64 字节。DiskINode 中记录了文件类型、文件地址映射关系以及文件访问信息等文件信息。

文件区占用 64-81919 号盘块。目录文件数据块、大文件的索引块以及普通文件数据块都存放其中，每一个数据块即对应一个 512 字节的盘块。

```

/*
 * 文件系统类(FileSystem)管理文件存储设备中
 * 的各类存储资源，磁盘块、外存INode的分配、
 * 释放。
 */
class FileSystem
{
public:
    /* static consts */
    static const int SUPER_BLOCK_SECTOR_NUMBER = 0;
    /* 定义SuperBlock位于磁盘上的扇区号，占据0，1两个扇区。 */

    static const int ROOTINO = 1;    /* 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，每个磁盘块可以存放512/64 = 8个外存Inode */
    static const int INODE_ZONE_START_SECTOR = 2;
    /* 外存Inode区位于磁盘上的起始扇区号 */
    static const int INODE_ZONE_SIZE = 64 - 2;
    /* 磁盘上外存Inode区占据的扇区数 */

    static const int DATA_ZONE_START_SECTOR = 64;
    /* 数据区的起始扇区号 */
    static const int DATA_ZONE_END_SECTOR = 81920 - 1;
    /* 数据区的结束扇区号 */
    static const int DATA_ZONE_SIZE = 81920 - DATA_ZONE_START_SECTOR;
    /* 数据区占据的扇区数量 */
}
    
```

3.2.2. SuperBlock 超级块结构

SuperBlock 区共 1024 个字节，占用两个扇区，负责通过栈进行 Inode 区空闲 Inode 的管理和文件区空闲数据盘块的管理。FileSystem 初始化时会从虚拟磁盘文件中读入并装载 SuperBlock 类。

对于空闲 Inode 节点的管理，SuperBlock 中的 s_isize 字段表示存储设备上的 Inode 区占用的总盘块数，在 SuperBlock 类构造时填入。s_inode[100]记录当前空闲的 Inode 号，s_ninode 记录当前空闲的 Inode 个数。

对于文件区的空闲数据块管理，与 Inode 管理类似。s_fsize 表示文件区的盘块总数，也在 SuperBlock 类构造时填入。s_free[100]记录当前空闲的数据盘块号，s_nfree 记录当前空闲的文件区数据块。与 Inode 不同，超出 100 个空闲盘块，则需要另选组长板块进行额外的空闲盘块登记。

```
/*
 * 文件系统存储资源管理块 (Super Block) 的定义。
 */
class SuperBlock
{
    /* Functions */
public:
    /* Constructors */
    SuperBlock();
    /* Destructors */
    ~SuperBlock();

    /* Members */
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[100];    /* 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[100];   /* 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_fmod;         /* 内存中super block副本被修改标志 */
int    s_ronly;        /* 本文件系统只能读出 */
int    s_time;         /* 最近一次更新时间 */
int    padding[47];    /* 填充使SuperBlock块大小等于1024字节 */
};

```

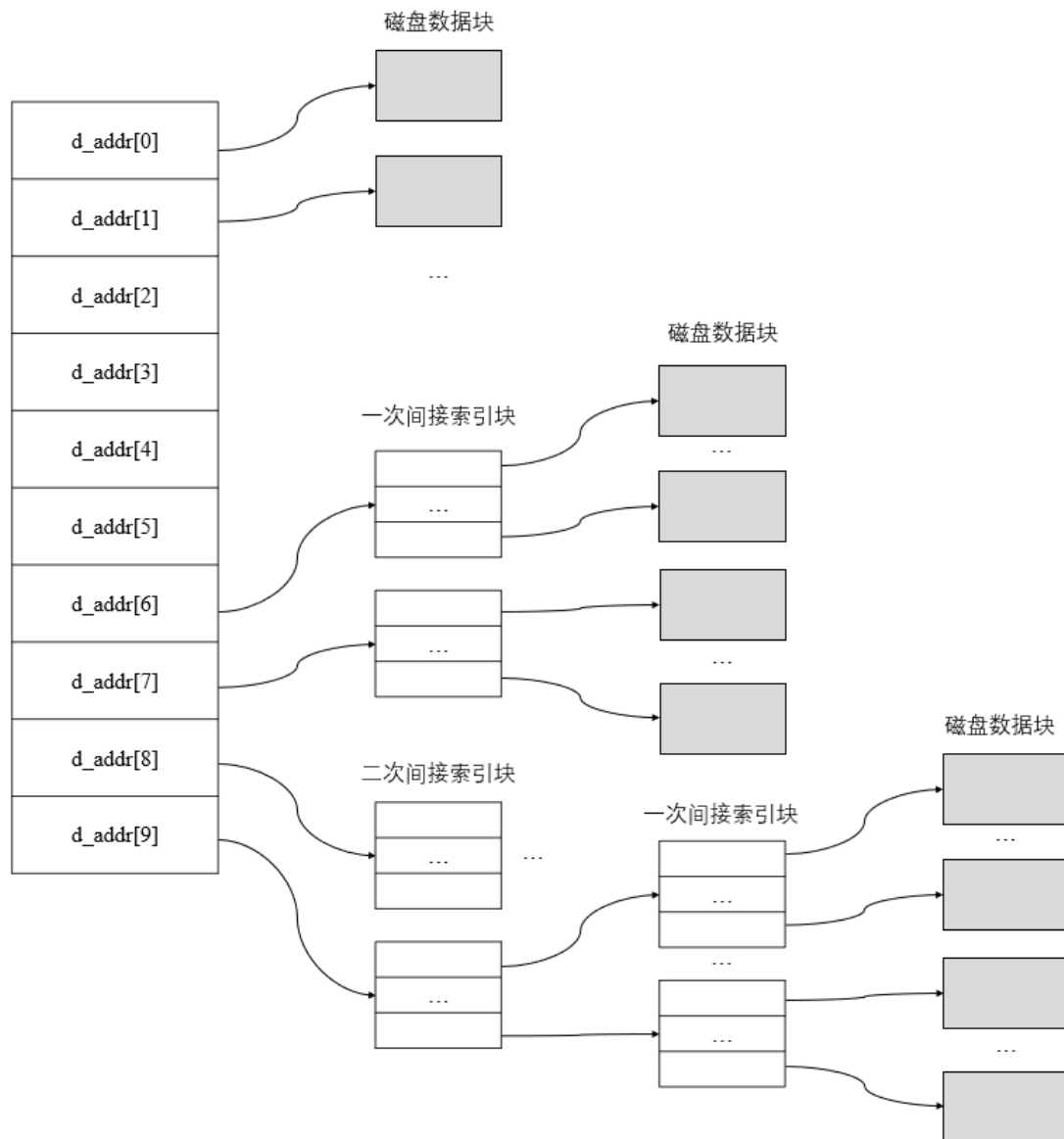
3.2.3. DiskInode 结构

该结构定义了磁盘 DiskInode 节点的存储格式，部分重要字段如下：

d_mode: 文件工作方式信息。在本次程序中，主要用到的是其第 13~14 位的文件类型编码（00 表示普通数据文件、10 表示目录文件），以及第 15 位的 IALLOC 标志表示当前 DiskInode 是否空闲。此外，第 0~8 位记录用户访问权，具体为“RWXRWXRWX”的格式。

d_size: 记录文件大小（以字节为单位）。

d_addr 数组: 记录文件各个数据盘块的索引地址。本次文件系统的索引实现与 UNIX V6++中保持一致。对于小型文件，d_addr 中前 6 个直接索引项直接指向对应的数据盘块号；对于大型文件，d_addr[6]、d_addr[7]指向一次间接索引块；对于巨型文件，d_addr[8]、d_addr[9] 这最后两个索引项为二次间接索引块。对于间接索引块来说，每个简介索引块表可以容纳 $512/4=8$ 个物理块号，可为 128 个文件的逻辑块与物理块建立对应关系。对应的索引树如下：



```

/*
 * 外存索引节点 (DiskINode) 的定义
 * 外存INode对象长度为64字节，
 * 每个磁盘块可以存放512/64 = 8个外存Inode
 */
class DiskInode
{
    /* Functions */

```

```
public:
    /* Constructors */
    DiskInode();
    /* Destructors */
    ~DiskInode();

    /* Members */
public:
    unsigned int d_mode; /* 状态的标志位, 定义见enum INodeFlag */
    int d_nlink; /* 文件在目录树中不同路径名的数量 */

    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_size; /* 文件大小, 字节为单位 */
    int d_addr[10]; /* 用于文件逻辑块好和物理块好转换的基本索引表 */

    int d_atime; /* 最后访问时间 */
    int d_mtime; /* 最后修改时间 */
};
```

3.2.4. DirectoryEntry 目录结构

目录文件同普通数据文件一样存储在文件区的盘块上。对于目录文件的盘块, 每个目录项 DirectoryEntry 共 32 字节, 其中 inode 号占 4 字节, 文件名占 28 字节。下图展示了根目录文件的存储格式示例:

根目录文件
(1#文件)

1# inode

...	
d_mode	10
d_uid	0
d_size	7*32
d_addr[0]	1024
d_addr其余	0

1024# 扇区 (数据块)

1	.\0
1	..\0
2	bin\0
3	etc\0
4	dev\0
5	home\0
6	shell\0

每个目录项32字节
文件名: 28字节
inode号: 4字节

注：本系统将指向父目录项的“..”与指向当前目录项的“.”显示地存入目录文件中，因此支持 cd 时的相对路径跳转。ls 展示目录时，可以通过“-a”参数显示以“.”开头的隐藏目录项。

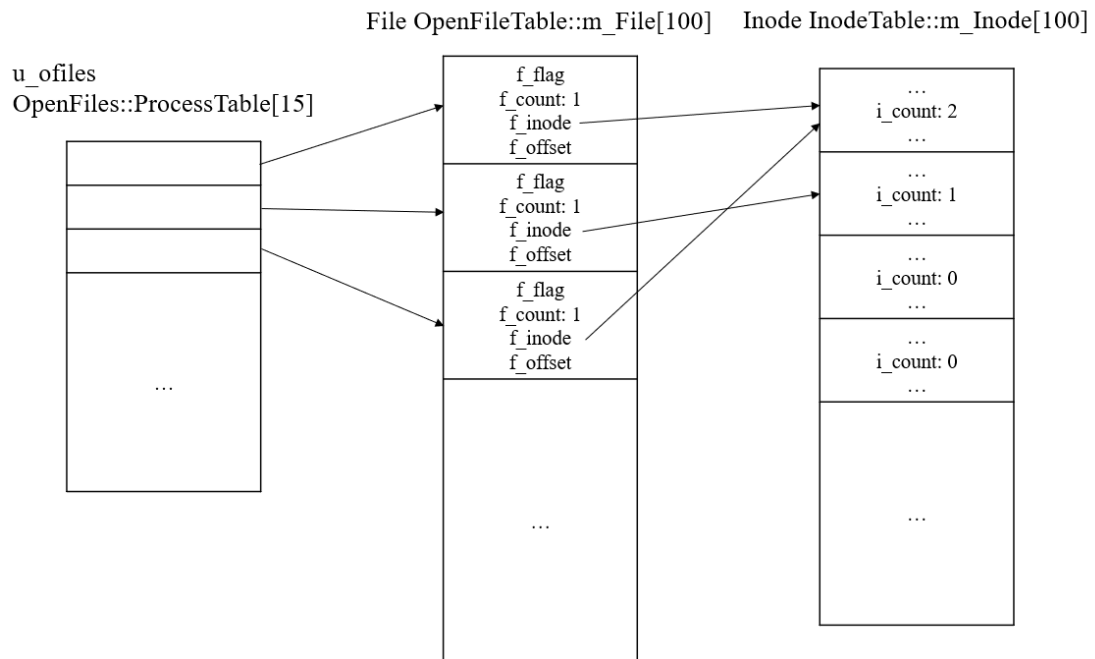
```
class DirectoryEntry
{
    /* static members */
public:
    static const int DIRSIZ = 28; /* 目录项中路径部分的最大字符串长度 */

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

    /* Members */
public:
    int m_ino;          /* 目录项中Inode编号部分 4字节*/
    char m_name[DIRSIZ]; /* 目录项中路径名部分 28字节*/
};
```

3.2.5. 打开文件结构

对于通过 fopen 打开的文件，系统会建立起进程打开文件列表、打开文件表、内存索引节点表的勾连关系，详见下页图。由于本次实验不涉及多进程，当前用户的打开文件表存储在其 User 结构中。每次打开文件时会分配一个独立的 File 控制块，并将 User 打开文件表中的对应表项指向该 File 结构。系统打开文件表 OpenFileTable 负责分配和管理打开文件对应的 File 对象，其记录了对打开文件的读写标识、文件读写指针等信息，并记录了指向对应文件 Inode 的指针。Inode 指针指向的 Inode 存储在内存索引节点表 InodeTable 中。



```

/*
 * 打开文件控制块File类。
 * 该结构记录了进程打开文件
 * 的读、写请求类型，文件读写位置等等。
 */
class File
{
public:
    /* Enumerate */
    enum FileFlags
    {
        FREAD = 0x1,          /* 读请求类型 */
        FWRITE = 0x2,         /* 写请求类型 */
        FPIPE = 0x4           /* 管道类型 */
    };

    /* 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 */
};

/*
 * 进程打开文件描述符表 (OpenFiles) 的定义
 * 进程的u结构中包含OpenFiles类的一个对象,
 * 维护了当前进程的所有打开文件。
 */
class OpenFiles
{
    /* static members */
public:
    static const int NOFILES = 15;    /* 进程允许打开的最大文件数 */

    /* Members */
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 */
};

```

3.2.6. 高速缓存块结构

Buf 类定义了本系统对缓存控制块的设计。相较于 Unix V6++, 由于本系统中不涉及到并发进程, 因此 Buffer 控制块的设计可以做诸多简化。例如, 原先的 B_WANTED、B_ASYNC、B_BUSY 等标志位可以取消, 统一设计为同步读写。具体来说, 标志位设计改进如下:

B_CLEAR: 所有标志位清空;

B_WRITE: 表示该缓存块待写入虚拟磁盘设备;

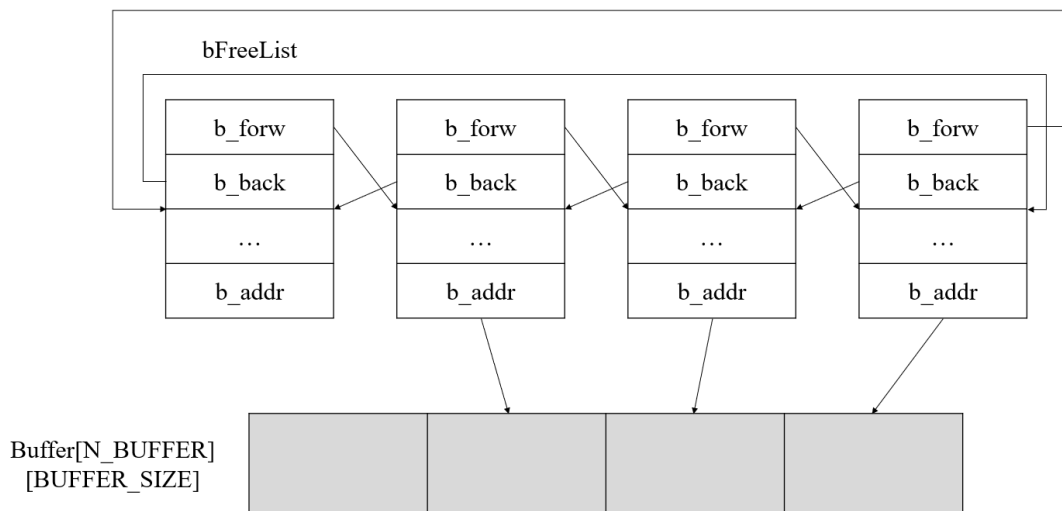
B_READ: 表示该缓存块待从虚拟磁盘设备读入;

B_DONE: 表示缓存块正确反映 (应) 存储在磁盘上的信息;

B_ERROR: 表示该缓存块读写虚拟磁盘失败;

B_DELWRI: 表示该缓存块延迟写。

BufferManager 类负责缓存块的管理与控制。每个缓存块对应一个上述的缓存控制块。由于本系统只涉及到一个设备，因此可以取消 Unix V6++原设计中的设备缓存队列，仅保留自由缓存队列进行缓存管理即可。示意图如下：



```
#define BUF_SIZE 512

/*
 * 缓存控制块buf定义
 * 记录了相应缓存的使用情况等信息；
 * 同时兼任I/O请求块，记录该缓存
 * 相关的I/O请求和执行结果。
 */
class Buf
{
public:
    enum BufFlag    /* b_flags中标志位 */
```

```

{
    B_CLEAR      = 0x0,      /* b_flags所有标记清空 */
    B_WRITE      = 0x1,      /* 写操作。将缓存中的信息写到硬盘上去 */
    B_READ       = 0x2,      /* 读操作。从盘读取信息到缓存中 */
    B_DONE       = 0x4,      /* I/O操作结束 */
    B_ERROR      = 0x8,      /* I/O因出错而终止 */
    B_DELWRI     = 0x10     /* 延迟写 */
};

public:
    unsigned int b_flags; /* 缓存控制块标志位 */

    Buf* b_forw;
    Buf* b_back;

    short  b_dev;          /* 主、次设备号 */
    int     b_wcount;      /* 需传送的字节数 */
    unsigned char* b_addr; /* 指向该缓存控制块所管理的缓冲区的首地址 */
    int     b_bufno;       /* 缓存块在缓存块队列中的下标 */
    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 */
};

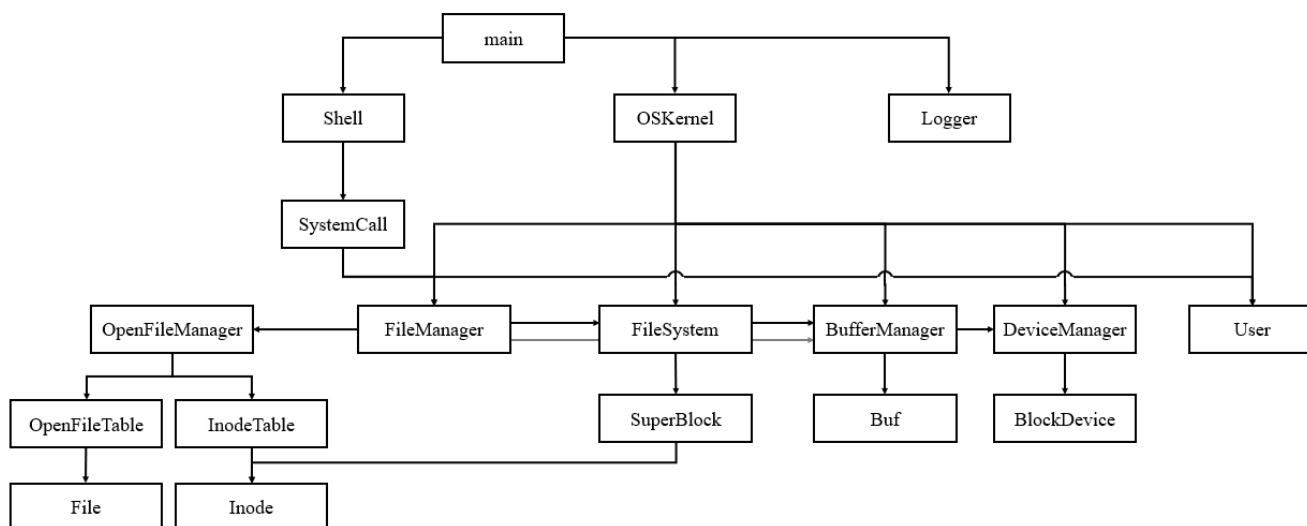
class BufferManager
{
public:
    /* static const member */
    static const int NBUF = 15;          /* 缓存控制块、缓冲区的数量 */
    static const int BUFFER_SIZE = BUF_SIZE; /* 缓冲区大小（字节） */

private:
    Buf bFreeList;                      /* 自由缓存队列控制块 */
    Buf m_Buf[NBUF];                   /* 缓存控制块数组 */
    unsigned char Buffer[NBUF][BUFFER_SIZE]; /* 缓冲区数组 */

    DeviceManager* m_DeviceManager;    /* 指向设备管理模块全局对象 */
};

```

3.3. 模块间的调用关系



3.4. 算法说明

本节对本系统实现的主要算法进行简述，具体流程详见 4.2 节。

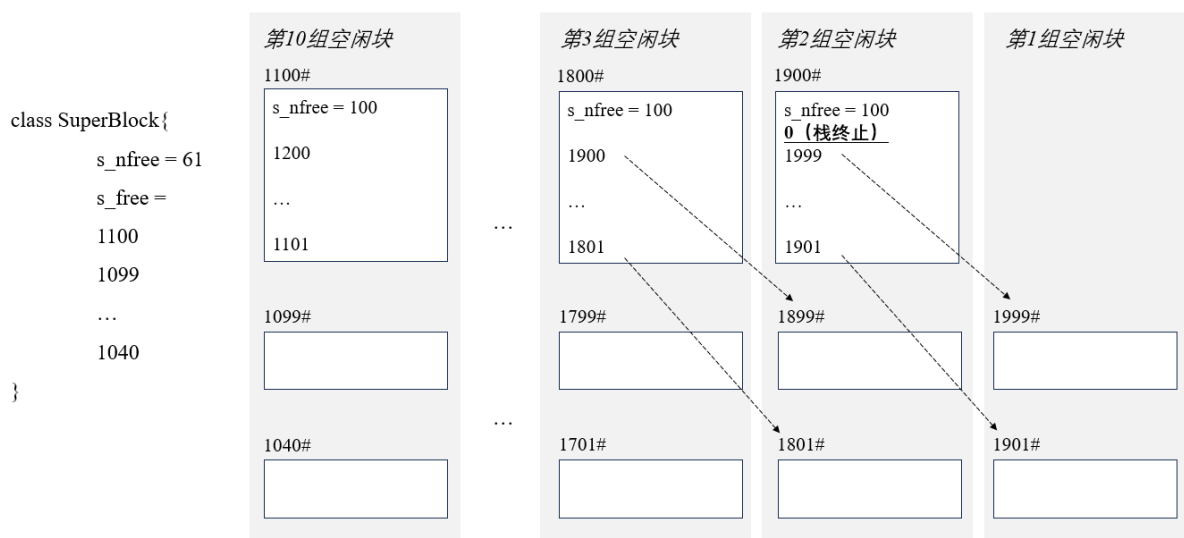
3.4.1. 高速缓存管理算法

本文件系统采用 LRU 算法进行高速缓存管理。LRU 即“最近最久未使用”（Least Recently Used）的缩写，其根据最近一段时间内最少被使用的缓存项进行淘汰。具体来说，通过 bFreeList 这一循环链表实现最近最少使用地缓存块分配与回收。分配缓存时，NotAvail 函数将缓存块从缓存队列首部取出；回收缓存块时，Brelse 函数将缓存块重新送入自由缓存队列的尾部。LRU 算法能够保证最近被释放的缓存块出现在队列尾部，而新取出（即队首）的高速缓存块是被释放最久的，即能够保证每次取出最近最久未使用的缓存块。

3.4.2. Inode 节点分配回收算法

Inode 节点通过栈式管理进行分配与回收。s_ninode 字段记录了当前超级块中记录的空闲 Inode 数量，s_inode 则作为栈存储每个空闲 Inode 节点的编号。在释放 Inode 节点时，若 SuperBlock 索引节点表中空闲 Inode 个数小于 100，则可以直接将 Inode 编号压栈。需要注意的是，由于 Inode 中 mode 的字段本身能够标识是否空闲，因此若 SuperBlock 登记的 Inode 满 100，则无需继续登记。反之，若当前 SuperBlock 中登记的所有 Inode 都已经被释放，则算法将遍历磁盘中的 Inode，找到 100 个空闲节点压入栈中进行登记。

3.4.3. 文件数据盘块分配回收算法



类似于 Inode 节点，文件数据盘块也通过栈式管理进行分配与回收。超级块最多能登记 100 个空闲块。不同于 Inode 的是，盘块无法记录是否空闲。对于盘块回收，若 s_free 栈不满，则数据块直接入栈；否则，则分配一个组长块记录当前的 s_free 信息，s_nfree 计为零重新登记。对于盘块分配，若栈内还有盘块则将当前栈顶弹出，否则需要重新装载组长块中记录的空闲盘块信息。

3.4.4. 目录项搜索算法

目录项搜索由 NameI 函数完成。首先，若目录由 “/” 开始，则从根目录开始搜索，否则从当前目录开始搜索。每次搜索在当前目录匹配与当前分量（下一个 “/” 前）相同的目录项，若有则进行下一段匹配，直到目录匹配完成。

3.4.5. 数据库索引映射算法

对于小型文件，则直接在索引表（i_addr[0] ~ i_addr[5]）得到数据块的直接索引；对于大型文件，则 i_addr[6]、i_addr[7] 存储的是间接索引，通过其得到一级间接索引表，再通过索引表中的信息得到盘块号；对于巨型文件，则首先从 i_addr[8]、i_addr[9] 中的索引得到二级间接索引表，再通过二级间接索引表得到一级间接索引表，便能够类似地通过索引表中的信息得到盘块号。

4. 详细设计

4.1. 重点函数与重点变量

4.1.1. VirtualFileDevice

重点函数/变量	说明
SECTOR_SIZE	每个扇区的大小
DEVICE_MEMORY	一级文件的总大小
NSECTOR	一级文件中扇区的个数
int Strategy(Buf* bp)	进行虚拟磁盘设备读/写操作
string filename	一级文件的名称
fstream file	一级文件的文件流
int Bno2Addr(int bno)	将物理盘块号转换为读写文件地址

int Read(Buf* bp)	读一级文件
int Write(Buf* bp)	写一级文件

4.1.2. BufferManager

重点函数/变量	说明
NBUF	缓存控制块和缓冲区的数量
BUFFER_SIZE	缓冲区大小，以字节为单位
Buf* GetBlk(short dev, int blkno)	申请一块缓存，用于读写设备 dev 上的字符块 blkno
void Brelse(Buf* bp)	释放缓存控制块 buf
Buf* Bread(short dev, int blkno)	读取一个磁盘块，dev 为设备号，blkno 为逻辑块号
void Bwrite(Buf* bp)	写入一个磁盘块
void Bawrite(Buf* bp)	异步写（实际上本系统中同步实现）
void Bdwrite(Buf* bp)	延迟写（打上 B_DELWRI，稍后写）
void ClrBuf(Buf* bp)	清空缓冲区内容
void Bflush(short dev)	将指定设备队列中延迟写的缓存全部输出到磁盘
Buf& GetBFreeList()	获取自由缓存队列控制块 Buf 对象引用
void NotAvail(Buf* bp)	从自由队列中移除指定的缓存控制块
Buf bFreeList	自由缓存队列控制块
Buf m_Buf[NBUF]	缓存控制块数组
unsigned char Buffer[NBUF][BUFFER_SIZE]	缓冲区数组

4.1.3. Buf

重点函数/变量	说明
enum BufFlag	缓存块标志位
Buf* b_forw	自由缓存队列后继缓存块指针
Buf* b_back	自由缓存队列前驱缓存块指针

4.1.4. FileManager

重点函数/变量	说明
enum DirectorySearchMode	目录搜索模式，用于 NameI()函数
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* pNode, 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 Stat()	Stat()获取文件信息
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
Inode* NameI(char (*func)(), enum DirectorySearchMode mode)	目录搜索，将路径转化为相应的 Inode
static char NextChar()	获取路径中的下一个字符
Inode* MakNode	被 Creat()系统调用使用，为创建新文件分配内核资源

(unsigned int mode)	
void MkNod()	创建特殊系统文件，这里用来建目录
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
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
Inode* rootDirInode	根目录内存 Inode

4.1.5. OpenFileTable

重点函数/变量	说明
NFILE	打开文件控制块 File 结构的数量
File* FAlloc()	在系统打开文件表中分配一个空闲的 File 结构
void CloseF(File* pFile)	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
File m_File[NFILE]	系统打开文件表，本系统中只有一个

4.1.6. InodeTable

重点函数/变量	说明
NINODE	内存 Inode 的数量
Inode* IGet(short dev, int inumber)	根据指定设备号 dev，外存 Inode 编号获取对应 Inode。

void IPut(Inode* pNode)	减少该内存 Inode 的引用计数
void UpdateInodeTable()	将所有被修改过的内存 Inode 更新到对应外存 Inode 中
int IsLoaded(short dev, int inumber)	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
Inode* GetFreeInode()	在内存 Inode 表中寻找一个空闲的内存 Inode
Inode m_Inode[NINODE]	内存 Inode 数组，每个打开文件都会占用一个内存 Inode

4.1.7. Inode

重点函数/变量	说明
InodeFlag	内存索引节点的标志位，包括 ILOCK、IUPD、IACC、IMOUNT、IWANT、ITEXT
IALLOC	文件被使用
IFMT	文件类型掩码
IFDIR	文件类型：目录文件
BLOCK_SIZE	文件逻辑块大小: 512 字节
ADDRESS_PER_INDEX_BLOCK	每个间接索引表(或索引块)包含的物理盘块号
SMALL_FILE_BLOCK	小型文件：直接索引表最多可寻址的逻辑块号
LARGE_FILE_BLOCK	大型文件：经一次间接索引表最多可寻址的逻辑块号
HUGE_FILE_BLOCK	巨型文件：经二次间接索引最大可寻址文件逻辑块号
void ReadI()	根据 Inode 对象中的物理磁盘块索引表，读取文件数据
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 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
unsigned int i_flag	状态的标志位
unsigned int i_mode	文件工作方式信息
int i_count	引用计数
int i_number	外存 inode 区中的编号
int i_size	文件大小，字节为单位
int i_addr[10]	用于文件逻辑块号和物理块号转换的基本索引表

4.1.8. FileSystem

重点函数/变量	说明
SUPER_BLOCK_SECTOR_NUMBER	定义 SuperBlock 位于磁盘上的扇区号，占据 0, 1 两个扇区。
ROOTINO	文件系统根目录外存 Inode 编号
INODE_NUMBER_PER_SECTOR	每个磁盘块可以存放的外存 Inode 数量
INODE_ZONE_START_SECTOR	外存 Inode 区位于磁盘上的起始扇区号
INODE_ZONE_SIZE	磁盘上外存 Inode 区占据的扇区数
DATA_ZONE_START_SECTOR	数据区的起始扇区号
DATA_ZONE_END_SECTOR	数据区的结束扇区号
DATA_ZONE_SIZE	数据区占据的扇区数量
void FormatDisk()	格式化磁盘
void LoadSuperBlock()	系统初始化时读入 SuperBlock
SuperBlock* GetFS(short dev)	获取该文件系统的 SuperBlock

void Update()	将 SuperBlock 对象的内存副本更新到存储设备
Inode* IAlloc(short dev)	在存储设备 dev 上分配一个空闲外存 Inode
void IFree(short dev, int number)	释放存储设备 dev 上编号为 number 的外存 Inode
Buf* Alloc(short dev)	在存储设备 dev 上分配空闲磁盘块
void Free(short dev, int blkno)	释放存储设备 dev 上编号为 blkno 的磁盘块

4.1.9. SuperBlock

重点函数/变量	说明
int s_nfree	直接管理的空闲文件数据盘块数量
int s_free[MAX_NUMBER_FREE]	直接管理的空闲文件数据盘块索引表
int s_ninode	直接管理的空闲外存 Inode 数量
int s_inode[MAX_NUMBER_INODE]	直接管理的空闲外存 Inode 索引表

4.1.10. User

重点函数/变量	说明
u_ar0[1]	存放系统调用的返回值给用户程序
u_arg[5]	存放当前系统调用参数
u_dirp	系统调用参数(一般用于 Pathname)的指针
u_cdir	指向当前目录的 Inode 指针
u_pdir	指向父目录的 Inode 指针

u_dent	当前目录的目录项
u_dbuf	当前路径分量
u_curdir	当前工作目录完整路径
u_error	存放错误码
u_ofiles	进程打开文件描述符表对象
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

4.1.11.OSKernel

重点函数/变量	说明
instance	OSKernel 单体类实例
m_BufferManager	缓存管理模块
m_DeviceManager	设备管理模块
m_FileSystem	文件系统
m_FileManager	文件管理模块
m_User	用户管理模块

4.1.12.SystemCall

重点函数/变量	说明
Sys_Open(const char* path, int mode)	打开文件
Sys_Close(int fd)	关闭文件
Sys_Read(int fd, unsigned char* buffer, int count)	读出文件

<code>Sys_Write(int fd, unsigned char* buffer, int count)</code>	写入文件
<code>Sys_MkNod(const char* path, int mode)</code>	新建文件夹
<code>Sys_ChDir(const char* path)</code>	改变当前目录
<code>Sys_Creat(const char* path, int mode)</code>	创建文件
<code>Sys_Seek(int fd, int offset, int mode)</code>	调整文件读写指针
<code>Sys_Unlink(const char* path)</code>	删除文件
<code>Sys_Stat(const char* path, const DiskInode* inode)</code>	读取 Inode 信息

4.2. 重点功能程序流程图

4.2.1. 高速缓存块管理

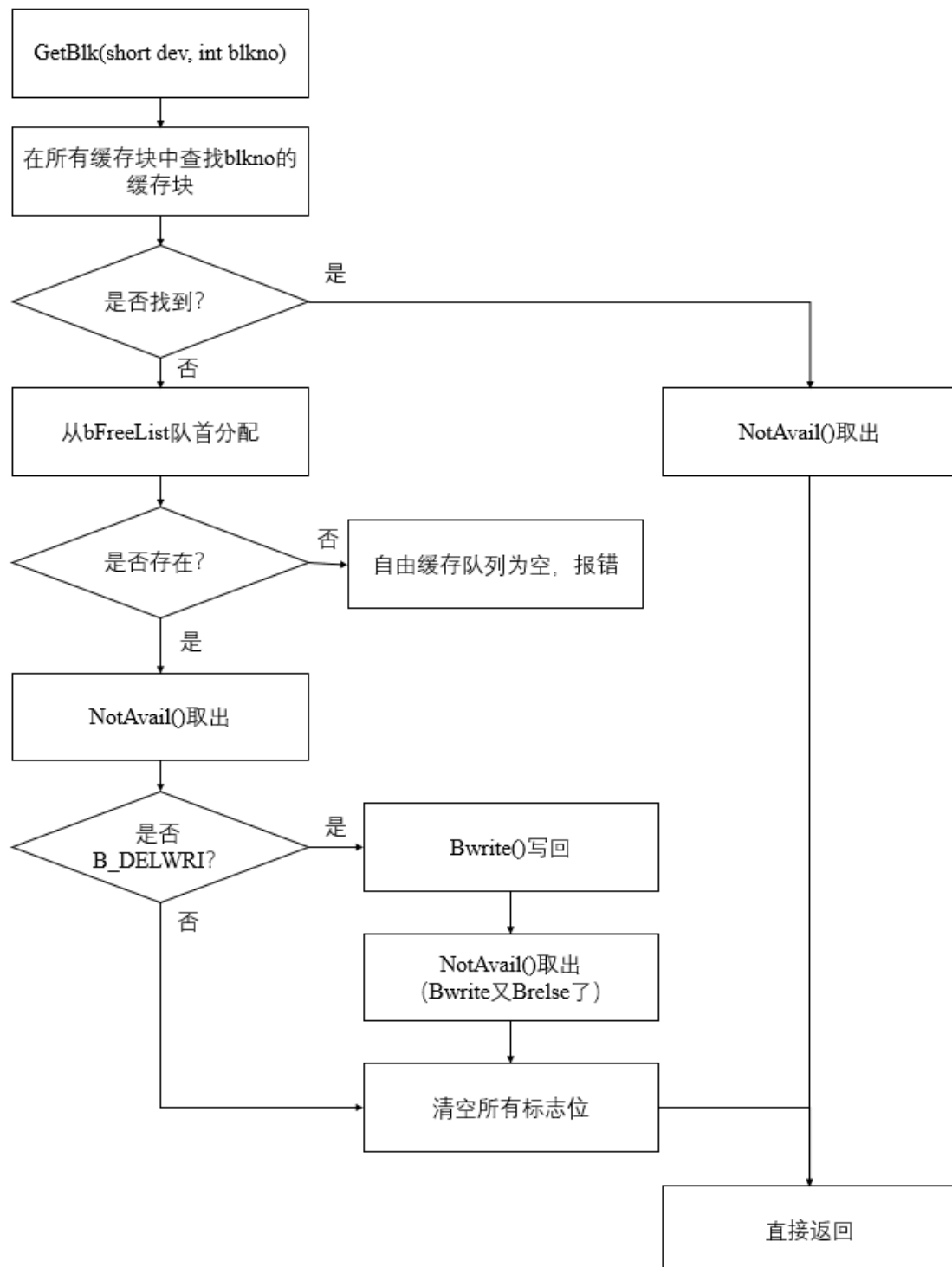
本文件系统采用 LRU 算法进行高速缓存管理，根据最近一段时间内最少被使用的缓存项进行淘汰。由于本系统中仅有一个设备，因此简化 Univ V6++ 中的设备队列，仅保留自由缓存队列。BufferManager 中记录 bFreeList 作为头节点保持不变，使用双向循环链表来实现 LRU 队列。

• 缓存块回收

缓存块回收通过 BufferManager 的 Brelse 函数实现。通过 Brelse 释放使用完毕的缓存块，将缓存块插入到 bFreeList 的最末尾。

• 缓存块分配

本算法在 Unix V6++ 的 GetBlk 算法上进行简化改进。首先寻找是否有可以复用的缓存块，若有则直接返回；否则则需从自由缓存队列中分配新缓存块。

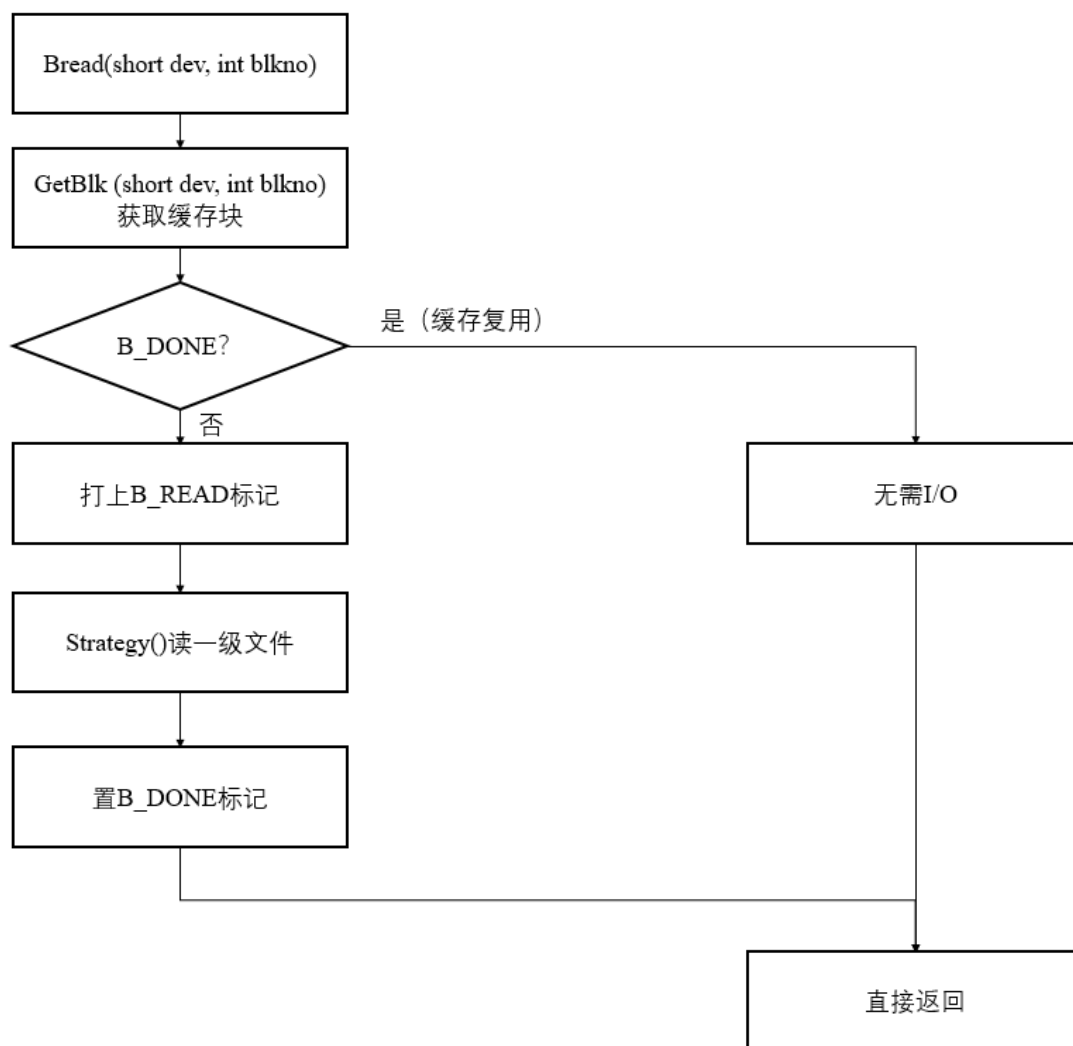


• 一级文件读写流程

对于文件写，采用延迟写。即未写满的 Buffer 在写回时只打上 `B_DELWRI`

的脏标记而不真正写回。当系统运行时，只有在缓存队列满（上图 GetBlk 自由缓存队列空时）才会进行一级文件写。另，通过 quit 退出系统，BufferManager 析构时，也会将所有的脏标记缓存块落盘。

对于文件读，有流程如下：

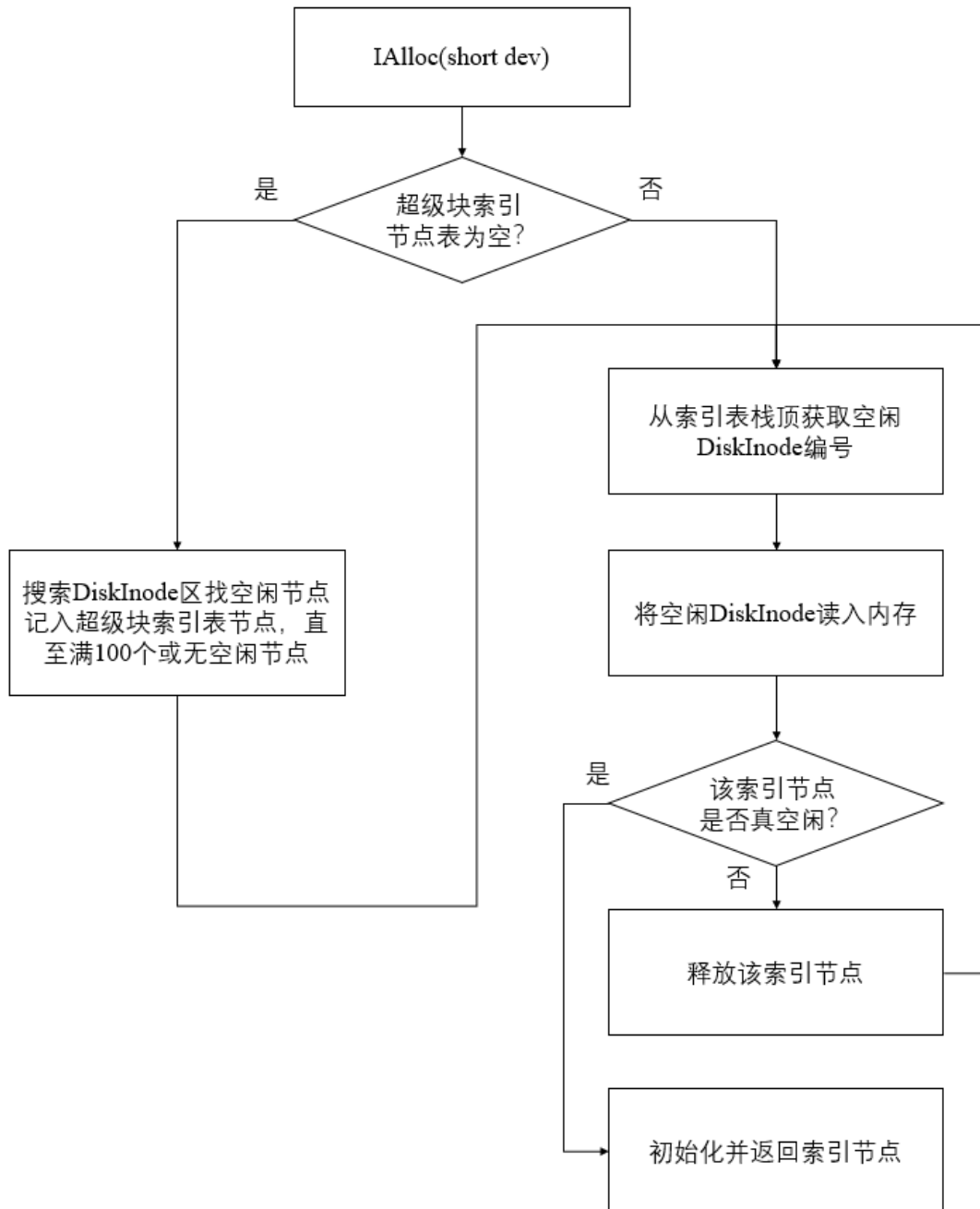


4.2.2. Inode 节点分配回收

• Inode 节点分配

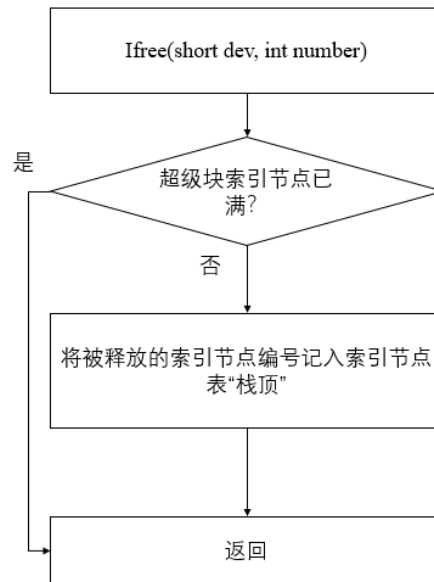
Diskinode 节点分配采用栈式管理。若 SuperBlock 中登记的栈不为空，则直

接分配；否则需要遍历一级文件的 Diskinode 区，将找到的空闲节点压栈。



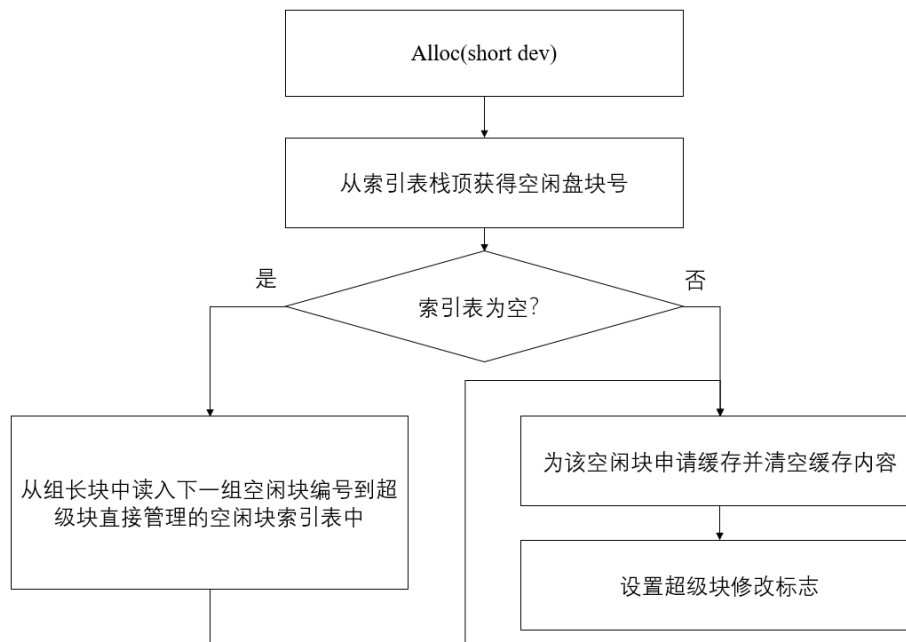
• Inode 节点回收

当释放一个 DiskINode 节点时，如果超级块索引节点表中空闲 DiskINode 看的个数小于 100，则将该索引节点编号记入栈顶；否则不进行操作。

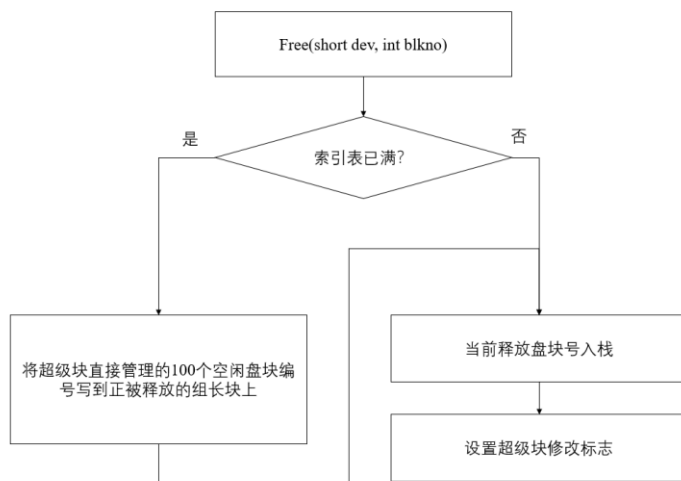


4.2.3. 文件数据盘块分配回收

• 文件数据盘块分配

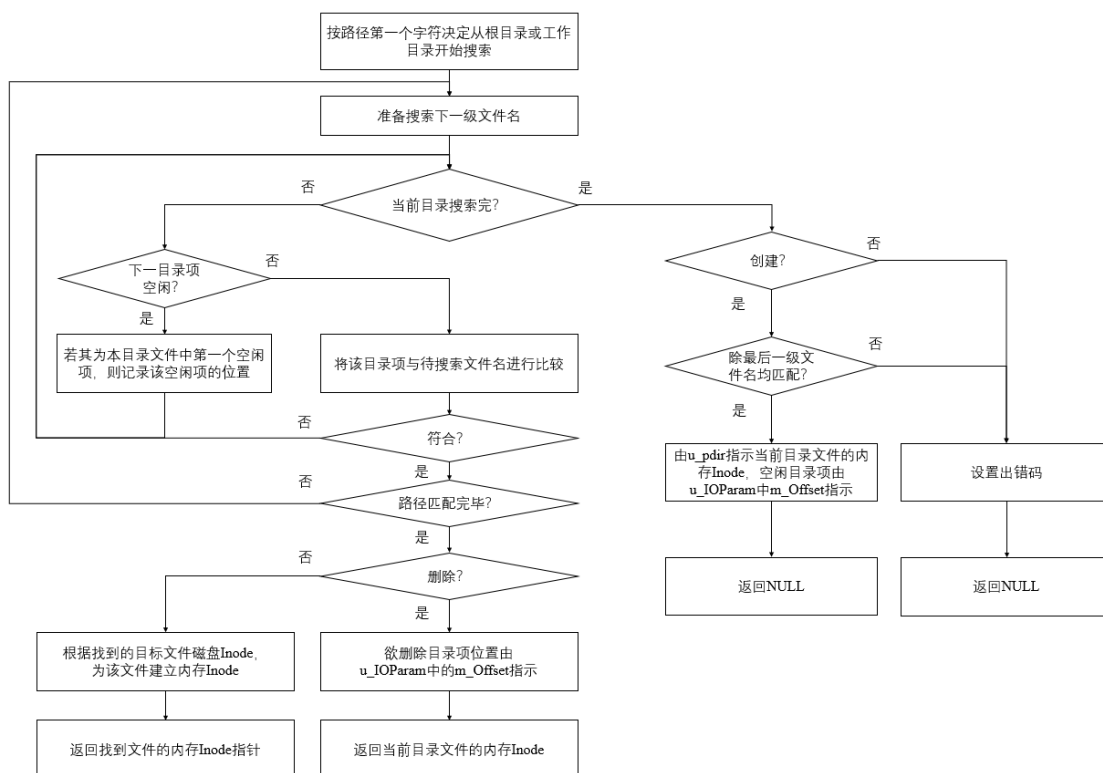


• 文件数据盘块回收



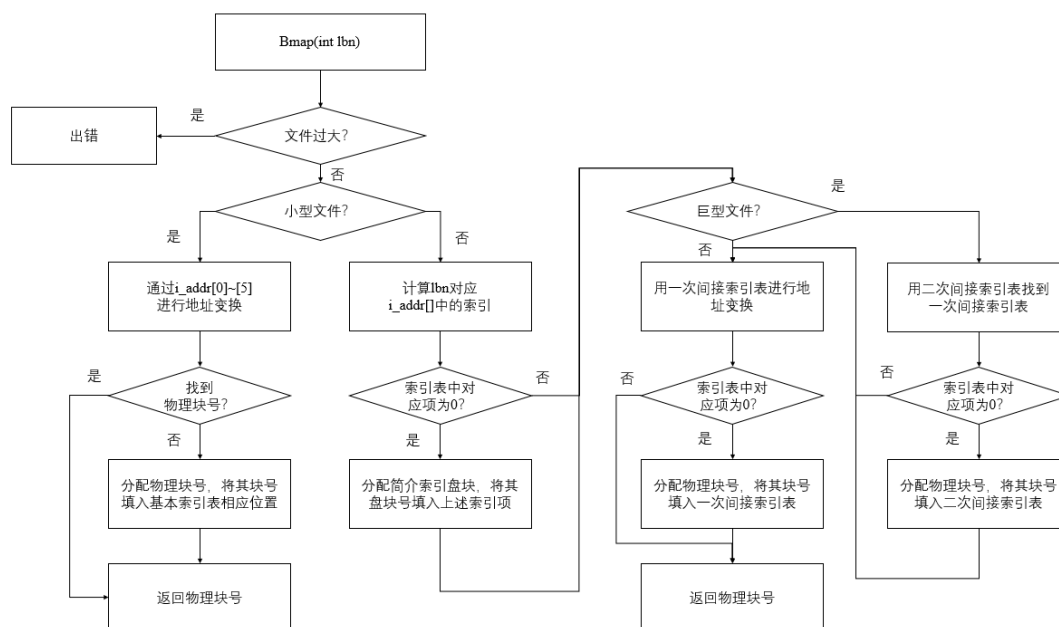
4.2.4. 目录项搜索

若目录由“/”开始，则 NameI 函数从根目录开始搜索，否则从当前目录开始。每次搜索在当前目录匹配与当前分量相同的目录项，直到目录匹配完成。

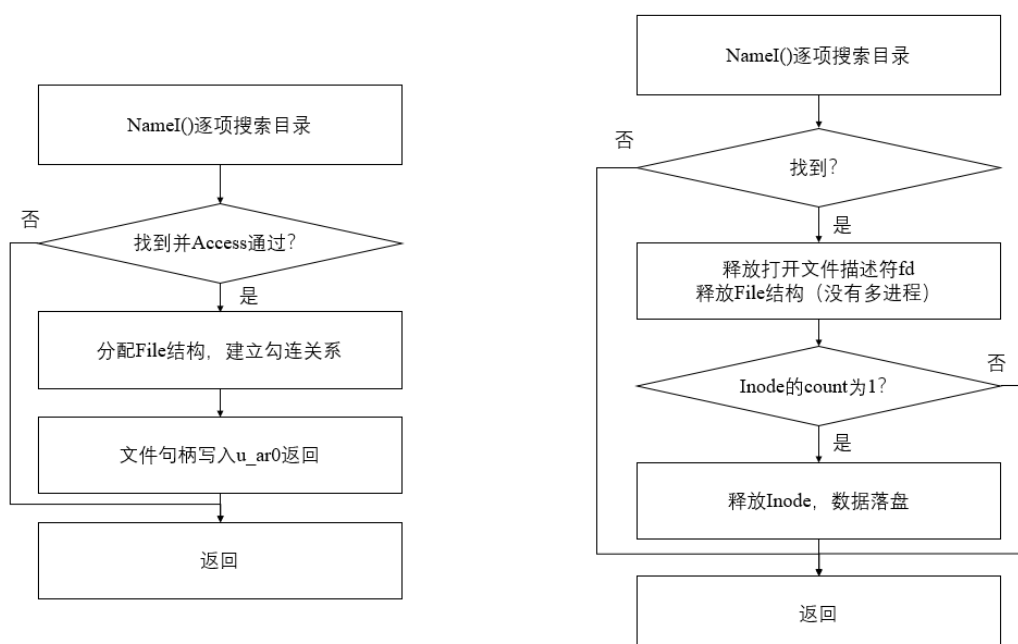


4.2.5. 数据块索引映射

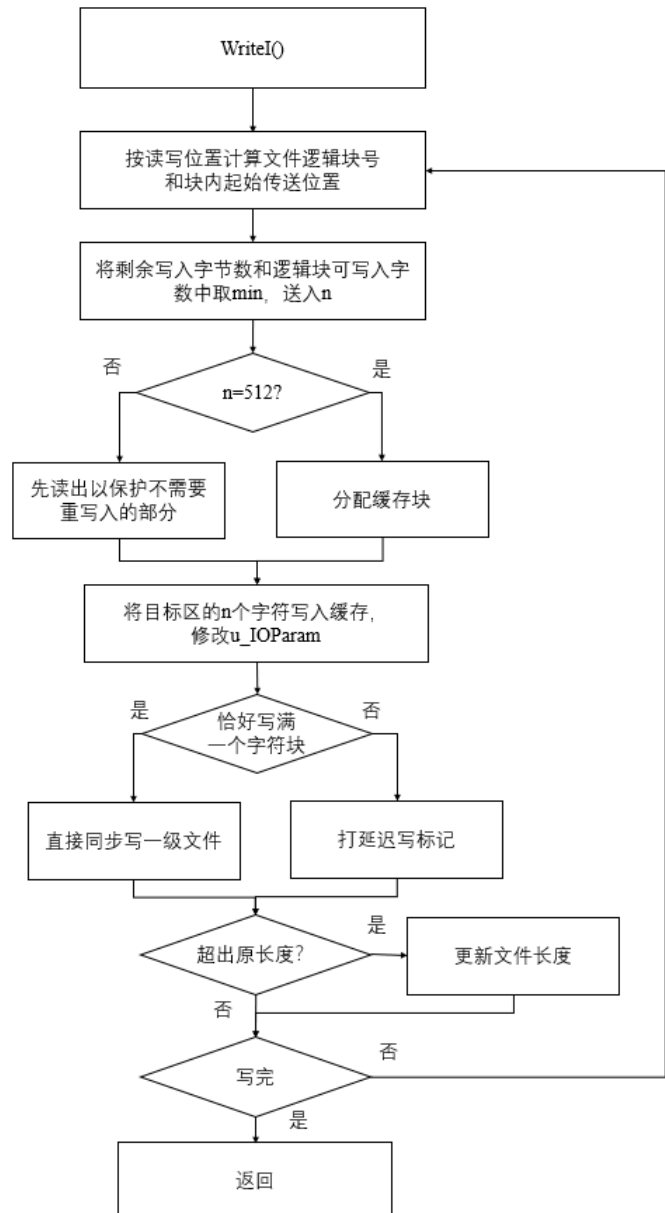
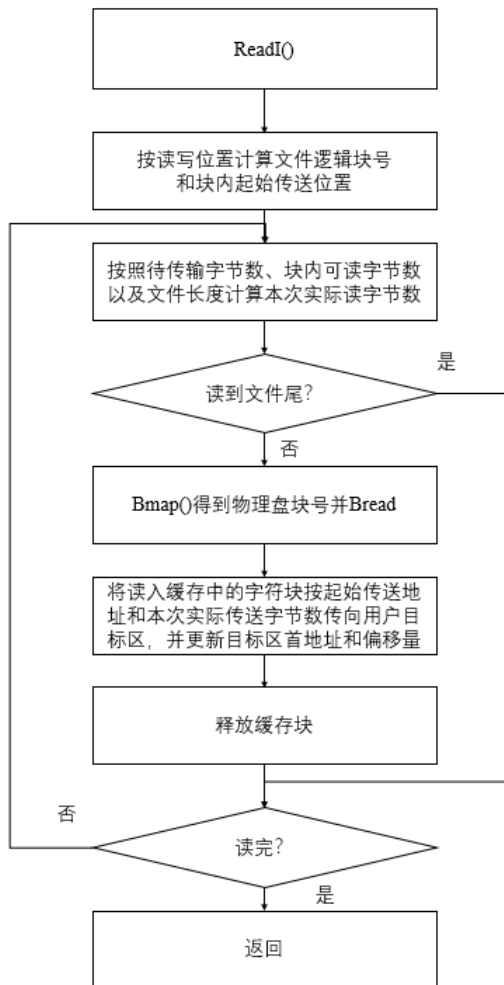
对于小型文件，则直接在基本索引表得到数据块的直接索引；对于大型文件，通过其得到一级间接索引表，再通过索引表中的信息得到盘块号；对于巨型文件，则首先得到二级间接索引表，再得到一级间接索引表从而映射。



4.2.6. 文件打开与关闭



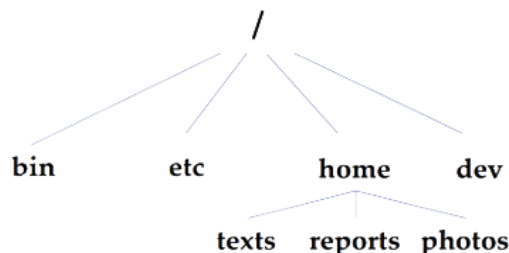
4.2.7. 文件读写流程



5. 运行结果分析

5.1. 程序运行结果展示

该部分进行课程设计要求的测试：



- 格式化文件卷；
- 用 `mkdir` 命令创建子目录，建立如图所示目录结构；
- 把课设报告，关于课程设计报告的 `ReadMe.txt` 和一张图片存进这个文件系统，分别放在 `/home/texts`，`/home/reports` 和 `/home/photos` 文件夹。

5.1.1. 格式化文件卷

```

C:\Users\BoyuanZheng\source x + -
/$ fformat
/$ |
  
```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	已解码的文本
00000000 3E 00 00 00 C0 3F 01 00 38 00 00 00 C7 3F 01 00	> ? . . 8 ? . .
00000010 C8 3F 01 00 C9 3F 01 00 CA 3F 01 00 CB 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000020 CC 3F 01 00 CD 3F 01 00 CE 3F 01 00 CF 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000030 D0 3F 01 00 D1 3F 01 00 D2 3F 01 00 D3 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000040 D4 3F 01 00 D5 3F 01 00 D6 3F 01 00 D7 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000050 D8 3F 01 00 D9 3F 01 00 DA 3F 01 00 DB 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000060 DC 3F 01 00 DD 3F 01 00 DE 3F 01 00 DF 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000070 E0 3F 01 00 E1 3F 01 00 E2 3F 01 00 E3 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000080 E4 3F 01 00 E5 3F 01 00 E6 3F 01 00 E7 3F 01 00	. ? . . . ? . . . ? . . . ? . .
00000090 E8 3F 01 00 E9 3F 01 00 EA 3F 01 00 EB 3F 01 00	. ? . . . ? . . . ? . . . ? . .
000000A0 EC 3F 01 00 ED 3F 01 00 EE 3F 01 00 EF 3F 01 00	. ? . . . ? . . . ? . . . ? . .
000000B0 F0 3F 01 00 F1 3F 01 00 F2 3F 01 00 F3 3F 01 00	. ? . . . ? . . . ? . . . ? . .
000000C0 F4 3F 01 00 F5 3F 01 00 F6 3F 01 00 F7 3F 01 00	. ? . . . ? . . . ? . . . ? . .
000000D0 F8 3F 01 00 F9 3F 01 00 FA 3F 01 00 FB 3F 01 00	. ? . . . ? . . . ? . . . ? . .
000000E0 FC 3F 01 00 FD 3F 01 00 FE 3F 01 00 FF 3F 01 00	. ? . . . ? . . . ? . . . ? . .
000000F0 9C 3F 01 00 9D 3F 01 00 9E 3F 01 00 9F 3F 01 00	. ? . . . ? . . . ? . . . ? . .

观察到 c.img 文件中的 SuperBlock 已经被正确格式化，空闲 Inode 和盘块也已经被正确记录在 SuperBlock。

```

027FFE00 01 00 00 00 2E 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE20 01 00 00 00 2E 2E 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFE90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFEA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFEB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFEC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFED0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFEE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFEF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFF90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFFA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFFB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFFC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFFD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFFE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .
027FFFF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . .

```

观察到系统中最后一个盘块被分配给根目录 rootInode 用作目录文件数据块，其第一、二个目录项分别记录“.”与“..”。

5.1.2. 用 mkdir 命令创建目录

使用 mkdir 指令依次创建所需的目录结构，并通过 tree 结构查看目录层级关系，发现目录结构符合题目要求。

```

C:\Users\BoyuanZheng\source > /$ mkdir bin
C:\Users\BoyuanZheng\source > /$ mkdir etc
C:\Users\BoyuanZheng\source > /$ mkdir home
C:\Users\BoyuanZheng\source > /$ mkdir dev
C:\Users\BoyuanZheng\source > /$ mkdir home/texts
C:\Users\BoyuanZheng\source > /$ mkdir home/reports
C:\Users\BoyuanZheng\source > /$ mkdir home/photos
C:\Users\BoyuanZheng\source > /$ tree
.
├── bin
├── etc
├── home
│   ├── texts
│   ├── reports
│   └── photos
└── dev
C:\Users\BoyuanZheng\source > /$ |
    
```

5.1.3. 存储文件

为测试文件系统，选择大文件进行存储。由于撰写报告时课程设计报告 pdf 还未生成，选用大小为 1928KB 的数学建模比赛论文作为 pdf；选用某项目的 ReadMe 文档作为 txt；选用大小为 4892KB 的图片作为测试 jpg。

基于 K-Means 聚类与判别分析的古埃及玻璃制品成分分析与鉴别

摘要

古埃及玻璃文物是丝绸之路沿线中西方贸易往来、文化交流的重要物证。我国玻璃文物材料上，吸收西亚与埃及地区的制陶技术，彰显了古代劳动人民的智慧。由于制陶材料与方法的差异，古代玻璃制品的化学成分不同，因此分为不同类型。在考古学研究中，玻璃文物的化学成分分析是其鉴别的重要手段。本文通过 K-Means 聚类分析、判别分析等方法，对玻璃文物的成分、颜色与玻璃文物的关系进行了研究。通过分析玻璃文物的化学成分、颜色与玻璃文物的关系，揭示了不同玻璃文物的化学成分与玻璃文物的关系。本文总结了不同玻璃文物的化学成分与玻璃文物的关系，并提出了相应的鉴别方法。

关键词：古埃及玻璃文物、化学成分、颜色、K-Means 聚类分析、判别分析

Gradient In COMRL

1. Prepare for the dataset

As collecting the dataset need a huge amount of time, we provide two datasets to valid as an example. You can download the datasets like [env_name].tar.bz2 [At Anonymous Site Here](https://drive.google.com/drive/folders/19KQqB1gTG2F0p82H_J-gt3NXeokRAS5?usp=sharing), and then extract them under the directory called **batch_data**.

```

...
--batch_data/
--AntDir-v0/
--data/
--seed_1_goal_2.62/
--obs.npy
--actions.npy
--next_obs.npy
--rewards.npy
--terminals.npy
--seed_2_goal_2.739/
...
--seed_40_goal_2.562/
...

```

To test the behaviour-ood performance, you need to download the datasets like [env_name].model.tar.bz2 [At Anonymous Site Here](https://drive.google.com/drive/folders/19KQqB1gTG2F0p82H_J-gt3NXeokRAS5?usp=sharing), and then extract them under the directory called **batch_data_copy**.

通过 cp 指令将外部的三个文件拷入系统之中，tree 查看文件结构如下：

```

C:\Users\BoyuanZheng\source x + v
/$ cp $report.pdf /home/reports/report.pdf
/$ cp $image.jpg /home/photos/image.jpg
/$ cp $README.md /home/texts/ReadMe.txt
/$ tree
.
├── bin
├── etc
├── home
│   ├── texts
│   │   └── ReadMe.txt
│   ├── reports
│   │   └── report.pdf
│   └── photos
│       └── image.jpg
└── dev

/$ |
    
```

通过 read 指令可以看出，文本文件存储正常：

```

C:\Users\BoyuanZheng\source x + v
└── dev

/$ fopen /home/texts/ReadMe.txt -r
fd: 0

/$ fread 0 500
# Gradient In COMRL

## 1. Prepare for the dataset
As collecting the dataset need a huge amount of time, we provide two datasets to valid
as an example. You can download the datasets like [env_name].tar.bz2 [At Anonymous Si
te Here](https://drive.google.com/drive/folders/19KQqbIgTG2F0p82M_J-gt3NXeokRASs?usp=
sharing), and then extract them under the directory called **batch_data**.
...
--batch_data/
  --AntDir-v0/
    --data/
      --seed_1_goal_2.62/
        --obs.npy

/$ fclose 0

/$ |
    
```


1. 文件读写指针位置为 0。因此可得，本次写的逻辑块号为 0，块内偏移地址为 0。本次写入字节数为 512；
2. 通过 Bmap 将逻辑地址转换为物理地址。在 `i_addr[0]` 中发现其对应的物理盘块号为 0，因此需要分配新磁盘块写入。将分配到的新盘块号计入 Jerry 文件对应 Inode 的 `i_addr[0]` 处；
3. 写入的数据为 512 字节数，因此无需先读入对应的磁盘块数据；
4. 将字符串中的前 512 字节写入到对应缓存块的 0~511 字节；
5. 修改读写指针为 512，刚好写满当前缓存块，执行写操作将缓存落盘；
6. 更新文件长度为 512 字节。

写第二个盘块：

1. 文件读写指针位置为 512。因此可得，本次写的逻辑块号 1，块内偏移地址为 0。本次写入字节数为 288；
2. 通过 Bmap 将逻辑地址转换为物理地址。在 `i_addr[1]` 中发现其对应的物理盘块号为 0，因此需要分配新磁盘块写入。将分配到的新盘块号计入 Jerry 文件对应 Inode 的 `i_addr[1]` 处；
3. 写入的数据为 $288 < 512$ 字节，未写满当前缓存块。因此需要先将物理盘块读入到新申请的高速缓存块防止数据丢失；
4. 将字符串中剩余的 288 字节写入到对应缓存块的 0~287 字节；
5. 修改读写指针为 800，并未写满当前缓存块。因此置缓存控制块的延迟写 `B_DELWRI` 标记，并将此缓存块直接释放；
6. 更新文件长度为 800 字节。

- 将文件读写指针定位到第 500 字节，读出 500 个字节到字符串 abc。

```
/$ flseek 0 500 -b  
/$ fread 0 500 >> abc
```

结果分析：首先将读写指针位置调整为 500，然后进行两次写盘块如下：

读第一个盘块：

1. 文件读写指针位置为 500。因此可得，本次读的逻辑块号为 0，块内偏移地址为 0。本次读入字节数为 12；
2. 通过 Bmap 将逻辑地址转换为物理地址。在 i_addr[0]中发现其对应的物理盘块号（即刚才分配的第一个盘块）；
3. 对该物理盘块申请缓存（复用刚才的缓存块，无需 I/O 读一级文件）；
4. 将该盘块内的 500~511 字节移入用户给定的缓冲区；
5. 释放缓存，修改文件读写指针位置到 512。

读第二个盘块：

1. 文件读写指针位置为 512。因此可得，本次读的逻辑块号为 1，块内偏移地址为 0。本次读入字节数为 288（到文件尾）；
2. 通过 Bmap 将逻辑地址转换为物理地址。在 i_addr[1]中发现其对应的物理盘块号（即刚才分配的第二个盘块）；
3. 对该物理盘块申请缓存（复用刚才的缓存块，无需 I/O 读一级文件）；
4. 将该盘块内的 0~288 字节移入用户给定的缓冲区；
5. 释放缓存，修改文件读写指针位置到 800。

- 将 abc 写回文件

```
/$ fwrite 0 << abc
```

结果分析：涉及到两个盘块的写，分两次，具体过程如下：

写第一个盘块：

1. 文件读写指针位置为 800。因此可得，本次写的逻辑块号为 1，块内偏移地址为 288。本次写入字节数为 224；
2. 通过 Bmap 将逻辑地址转换为物理地址。在 `i_addr[0]` 中发现其对应的物理盘块号（即刚才分配的第二个盘块）；
3. 写入的数据为 $224 < 512$ 字节，因此无需要读入对应的磁盘块数据（可以复用在自由缓存队列中的 Buffer，无需从一级文件读）；
4. 将字符串中的前 224 字节写入到对应缓存块的 288~511 字节；
5. 修改读写指针为 1024，刚好写满当前缓存块，执行写操作将缓存落盘。

写第二个盘块：

1. 文件读写指针位置为 1024。因此可得，本次写的逻辑块号 2，块内偏移地址为 0。本次写入字节数为 76；
2. 通过 Bmap 将逻辑地址转换为物理地址。在 `i_addr[2]` 中发现其对应的物理盘块号为 0，因此需要分配新磁盘块写入。将分配到的新盘块号计入 Jerry 文件对应 Inode 的 `i_addr[2]` 处；
3. 写入的数据为 $288 < 512$ 字节，未写满当前缓存块。因此需要先将物理盘块读入到新申请的高速缓存块防止数据丢失；
4. 将字符串中剩余的 76 字节写入到对应缓存块的 0~75 字节；

5. 修改读写指针为 1100，并未写满当前缓存块。因此置缓存控制块的延迟写 B_DELWRI 标记，并将此缓存块直接释放；

6. 更新文件长度为 1100 字节。

5.3. 测试其他命令及输出

部分指令如 mkdir、flseek 等已经在前文测试中展示过，此处不再赘述。

5.3.1. help

```
/$ help
ls      <列出路径下所有文件>
cd      <切换当前目录>
cp      <复制文件(支持文件系统内外复制)>
mv      <移动文件>
mkdir   <新建目录项>
fformat <格式化磁盘>
fcreat  <新建文件>
fdelete <删除文件>
fopen   <打开文件>
fclose  <关闭文件>
fread   <读文件>
fwrite  <写文件>
flseek  <调整文件读写指针>
tree    <展示目录树>
quit    <退出系统>
```

5.3.2. ls

若指令格式不正确，则指令会打印出提示信息。ls path 能够打印出目录下的所有目录项，但不包括以“.”开头的隐藏项。使用“-a”模式能够列举出包含隐藏项在内的所有目录项。

```
/$ ls -?
ls <列出路径下所有文件>
Usage:  ls (current path)
        ls <path>
        -a 全部展示

/$ ls
bin      etc      home     dev      test

/$ ls -a
.        ..       bin      etc      home     dev      test
```


5.3.3. cd

cd 指令用于切换目录。支持以无前缀、“.”、“..”跳转的相对路径（通过目录项插入对应项实现，无需特别判断），也支持“/”开头的绝对路径。

```
/$ cd home
/home$ cd ../
/$ cd home
/home$ cd ./photos
/home/photos$ cd /etc
/etc$ |
```

cd 指令会对目录路径进行合法性检查并给出错误提示：

```
/etc$ cd text.txt
非目录路径

/etc$ cd abc
文件或目录不存在
```

5.3.4. cp

cp 指令支持文件系统内外复制，外部路径需在路径前加“\$”以示区分（前文中已经展示）。下图展示了文件系统内的复制：

```
/$ cp /home/texts/ReadMe.txt text.txt

/$ ls
bin      etc      home     dev      text.txt

/$ cp
cp <复制文件(支持文件系统内外复制)>
Usage:   cp <src> <dst>
         为表区分，请在外部路径前加 '$'
```

若源文件不存在，则给出错误处理如下：

```
/etc$ cp 1.txt 2.txt
src: 文件或目录不存在
```

5.3.5. mv

```
/$ ls
bin      etc      home     dev      text.txt

/$ mv text.txt /etc/text.txt

/$ ls
bin      etc      home     dev

/$ ls etc
text.txt
```

mv 指令与 cp 指令类似，只是将文件复制变为文件移动，如上例所示。

5.3.6. fopen

```
/$ fopen home/reports/report.pdf -r
fd: 0

/$ fopen home/reports/report.pdf
必须指定至少一种打开方式

/$ fopen home/reports/report.pdf -w
fd: 1
```

fopen 通过指定文件打开方式“-r”、“-w”来打开文件，也可以同时指定读写两种方式进行文件打开。若文件成功打开，则会返回文件句柄以供下一步操作（fread、fwrite）。若未指定打开方式，则 fopen 会给出错误提示如上图。

5.3.7. fdelete

```
/etc$ fdelete text.txt

/etc$ ls

/etc$ fdelete a.txt
文件或目录不存在

/etc$ fdelete
fdelete <删除文件>
Usage:  fdelete <path>
```

5.3.8. fread、fwrite

fread、fwrite 支持通过标准输入输出流进行读写，以及读出或写入字符串。前者在前文展示过，此处展示后者。

```
/$ fwrite 0 abcdefghijklmn

/$ fseek 0 0 -b

/$ fread 0 5
abcde

/$ fread
fread <读文件>
Usage:  fread <fd> <count> [>> <str> (写入到字符串中)]

/$ fwrite
fwrite <写文件>
Usage:  fwrite <fd> <str>          从标准输入流写入
        fwrite <fd> << <str>      从字符串中写入
```

5.3.9. tree

展示给定目录（默认当前文件夹）的目录结构。

```
/$ tree
.
├── bin
├── etc
├── home
│   ├── texts
│   │   └── ReadMe.txt
│   ├── reports
│   │   └── report.pdf
│   └── photos
│       └── image.jpg
└── dev
```

5.3.10.fformat

```
/$ fformat
/$ |
```

fformat 后文件系统会自动重启内核，并回到根目录下。

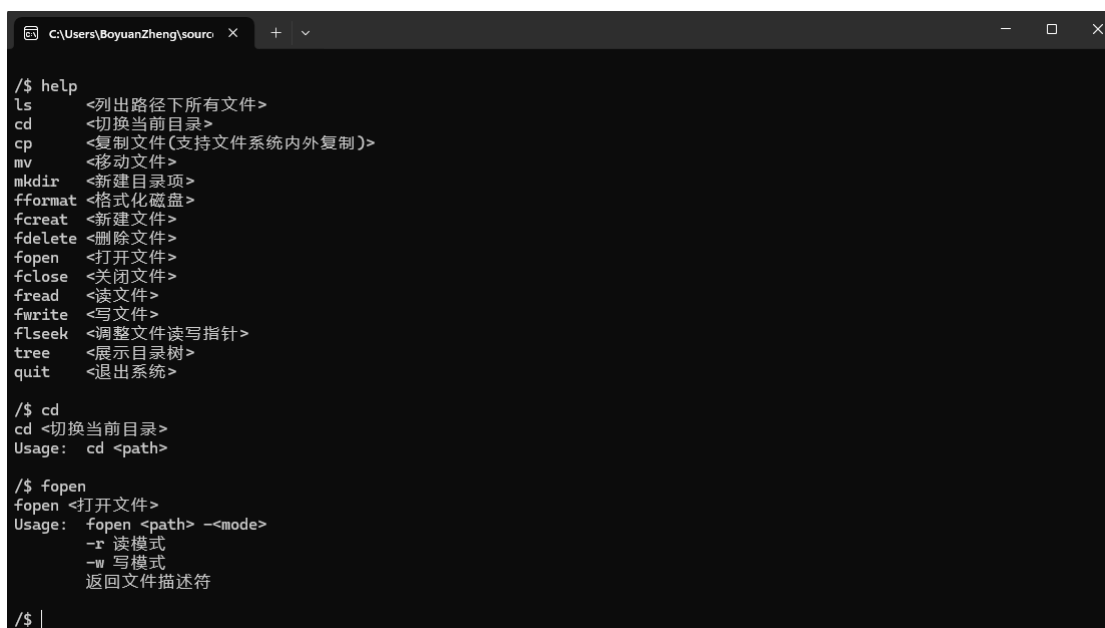
5.3.11.quit

退出文件系统。

6. 用户使用说明

6.1. 使用界面

用户通过命令行进行交互。输入 **help** 指令可以得到所有支持的命令格式。若命令不存在，则系统会给出对应的错误提示；若命令格式错误，则系统会给出该指令的具体操作格式以及各个参数的含义。



```

C:\Users\BoyuanZheng\source x + v
/$ help
ls      <列出路径下所有文件>
cd      <切换当前目录>
cp      <复制文件(支持文件系统内外复制)>
mv      <移动文件>
mkdir   <新建目录项>
fformat <格式化磁盘>
fcreat  <新建文件>
fdelete <删除文件>
fopen   <打开文件>
fclose  <关闭文件>
fread   <读文件>
fwrite  <写文件>
flseek  <调整文件读写指针>
tree    <展示目录树>
quit    <退出系统>

/$ cd
cd <切换当前目录>
Usage: cd <path>

/$ fopen
fopen <打开文件>
Usage: fopen <path> -<mode>
      -r 读模式
      -w 写模式
      返回文件描述符

/$ |
    
```

6.2. 注意事项

1. 初次使用系统必须使用 fformat 格式化建立文件系统；
2. 使用时需注意，退出系统必须通过“quit”指令，高速缓存队列中的脏缓存块才会通过 BFlush 函数落盘。否则，再次启动时可能导致系统崩溃。此时，可以通过 fformat 重新格式化系统，但会导致所有数据丢失。

7. 心得体会

这本次操作系统课程设计的时间过程中，我在 Unix V6++的基础上实现了一个类 UNIX 二级文件系统。在开始编写代码之前，我首先仔细地重新阅读了 Unix V6++的源代码，对其设计有了更深刻的理解。此后，我通过在实际调试过程中更加熟悉了 UNIX 文件系统的架构与算法，尤其是关于高速缓存的管理分配、Inode 与空闲盘块的分配等等算法。由于本实验只涉及到单进程、单用

户，我也自己创新性地做了不少改进与优化，可以说收获颇丰。在完成本次课程设计的过程中，我在实践中不断温习操作系统的基础知识，也通过不断地调试弥补了在操作系统代码实现层面的认知不足。当然，此次的二级文件系统还略显稚嫩，希望在此基础上我未来能够进一步完善多用户管理、多进程控制和多设备管理等进阶内容，不断提升自我、继续进步！

8. 参考文献

- [1] 方钰, 邓蓉, 陈闾中. 操作系统原理[M]. 上海:同济大学出版社
- [2] J. Lions, “莱昂氏 unix 源代码分析.”
[https://www.tsingfun.com/uploadfile/2015/1202/莱昂氏 unix 源代码分析.pdf](https://www.tsingfun.com/uploadfile/2015/1202/莱昂氏_unix_源代码分析.pdf).
- [3] 尤晋元, Unix 操作系统教程[M]. 西安:电子科技大学出版社, 1985 年 6 月