



操作系统·课程设计

2054435 廖偲宇

2023 年 5 月 20 日

目录

1	项目概述	1
1.1	项目内容	1
1.2	项目要求	1
1.3	报告要求	2
1.4	开发环境	3
2	概要设计	3
2.1	任务分解	3
2.2	结构说明	3
3	详细设计	6
3.1	重点函数与重点变量	6
3.2	实现流程	8
3.2.1	盘块申请	8
3.2.2	盘块释放	9
3.2.3	DiskInode 申请	9
3.2.4	DiskInode 释放	9
3.2.5	索引映射	9
3.2.6	目录查找	12
3.2.7	缓存替换	12
3.3	项目结构	14
4	测试与分析	14
4.1	简单命令测试	14
4.2	复杂命令测试 · 存放指定文件	17
4.3	复杂命令测试 · 文件写入定位	19
5	课设总结	21
6	参考资料	21

1 项目概述

1.1 项目内容

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++ 的一张磁盘。磁盘中存储的信息以块为单位，每块 512 字节。

具体来说，需要：

1. 实现对该逻辑磁盘的基本读写操作
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 ***
 - SuperBlock
 - 数据结构定义
 - 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
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
 - ...
 - fdelete

1.2 项目要求

课程设计的具体要求如下：

1. 完成前述所有数据结构与算法的设计
2. 实现一个控制台程序，图形界面或者命令行方式，等待用户输入，提供文件系统的基本功能，根据用户不同的输入，返回结果

1.3 报告要求

实验报告需包含以下内容：

1. 文件系统结构说明，至少包括：
 - SuperBlock 及 Inode 区所在位置及大小
 - Inode 节点数据结构的定义及索引结构的说明
 - SuperBlock 数据结构的定义及对 Inode 节点及文件数据区管理的相关算法
2. 目录结构说明，至少包括：
 - 目录文件的结构
 - 目录检索算法的设计
 - 目录结构增、删、改的设计
3. 文件打开结构说明，至少包括：
 - 文件打开结构的设计
 - 内存 Inode 节点数据结构的定义及分配与回收
 - 文件打开过程
4. 文件系统的实现，至少包括：
 - 文件读写操作的实现流程
 - 文件其他操作的实现流程
5. 高速缓存结构说明，至少包括：
 - 缓存控制块的设计
 - 缓存队列的设计及分配和回收算法
 - 借助缓存实现对一级文件的读写操作的流程
6. 此外，上述内容要求：
 - 数据结构定义
 - 重点函数的重点变量需说明，重点功能部分要绘制清晰的程序流程图，画出函数调用关系
7. 通过命令行方式完成下列操作：
 - 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
 - 把你的课设报告，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
8. 通过命令行方式测试：
 - 新建文件 /test/Jerry，打开该文件，任意写入 800 个字节
 - 将文件读写指针定位到第 500 字节，读出 500 个字节到字符串 abc

- 将 abc 写回文件

9. 上述内容要求

- 程序运行结果展示说明
- 测试命令及输出结果，结果分析

1.4 开发环境

课程项目完成过程的开发环境如下：

表 1: 开发环境

环境	描述	版本
集成开发环境	CLion	2022.1.3
编程语言	C++	C++14
构建系统工具	CMake	3.22

2 概要设计

2.1 任务分解

了解课设项目内容后，对其进行任务分解，将任务模块分解为以下几个部分：

- _Kernel

本系统的核心类 (内核)，实现系统调用模块。具体来说，实现了项目支持命令的具体操作的系统接口，负责 fformat、mkdir、cd、ls、fdelete、fcreat、fopen、fread、fwrite、fseek、fclose、exit、fmount 等系统功能的实现细节。

- FileSystem

本系统的文件系统类，实现文件管理模块。具体来说，负责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。

- FileManager

本系统的文件管理类，实现文件管理模块。具体来说，负责目录项的管理操作等。

- OpenFileManager

本系统的打开文件管理模块。具体来说，负责文件打开结构与内存 INode 表的管理。其中，INodeTable 负责内存 INode 结构的分配和释放；OpenFileTable 负责对打开文件实现文件描述符的分配，以及在文件关闭时对文件描述符进行回收。

2.2 结构说明

1. 文件系统结构

采用一个磁盘模拟文件系统，磁盘结构划分（文件系统结构）设计如下：

```
1      class FileSystem
2      {
3      public:
4          static const int NMOUNT = 5;
5
6          static const int SUPER_BLOCK_SECTOR_NUMBER = 200;
7
8          static const int ROOTINO = 0;
9          static const int INODE_NUMBER_PER_SECTOR = 8;
10         static const int INODE_ZONE_START_SECTOR = 202;
11         static const int INODE_ZONE_SIZE = 1024 - 202;
12
13         static const int DATA_ZONE_START_SECTOR = 1024;
14         static const int DATA_ZONE_END_SECTOR = 18000 - 1;
15         static const int DATA_ZONE_SIZE = 18000 - DATA_ZONE_START_SECTOR;
16     }
17
```

其中，定义超级块 SuperBlock 的起始扇区号为 200，占据磁盘上的 200、201 两个扇区；定义外存 Inode 的起始扇区号为 202，占据磁盘上的 202-1023（共 822 个）扇区；定义数据区 Data 的起始扇区号为 1024，占据磁盘上的 1024-17999（共 16976 个）扇区。

2. 超级块结构

文件系统存储资源管理块结构设计如下：

```
1      class SuperBlock
2      {
3      public:
4          int      s_isize;
5          int      s_fsize;
6          int      s_nfree;
7          int      s_free[100];
8          int      s_ninode;
9          int      s_inode[100];
10         int      s_fmod;
11         int      padding[51];
12     };
13
```

其中，s_isize、s_ninode、s_inode 负责管理 Inode 区；s_fsize、s_nfree、s_free 负责管理数据盘块；padding 数组用于填充使 SuperBlock 块大小等于 1024 字节，占据 2 个扇区。

3. 内存结构

内存 Inode 结构设计如下：

```

1      class Inode
2      {
3      public:
4          unsigned int i_flag;
5          unsigned int i_mode;
6
7          int         i_count;
8          int         i_nlink;
9          int         i_number;
10
11         int         i_size;
12         int         i_addr[10];
13     };
14

```

其中, `i_flag` 为状态的标志位, `i_mode` 为文件工作方式信息, `i_count` 为引用计数, `i_nlink` 为文件联结计数 (即该文件在目录树中不同路径名的数量), `i_number` 为外存 Inode 区中的编号, `i_size` 为文件大小 (以字节为单位), `i_addr` 数组为文件逻辑块号和物理块号转换的基本索引表, 用于逻辑盘块到物理盘块映射。

4. 外存结构

外存索引节点 `DiskInode` 结构设计如下:

```

1      class DiskInode
2      {
3      public:
4          unsigned int d_mode;
5          int         d_nlink;
6          int         d_size;
7          int         d_addr[10];
8          int         padding[3];
9     };
10

```

其中, `d_mode` 为状态的标志位, `d_nlink` 为文件联结计数 (即该文件在目录树中不同路径名的数量), `d_size` 为文件大小 (以字节为单位), `d_addr` 数组为用于文件逻辑块号和物理块号转换的基本索引表, 用于填充使得 `DiskInode` 类占 64 个字节。

5. 打开文件结构

打开文件 `File` 结构设计如下:

```

1      class File
2      {
3      public:

```

```

4         unsigned int f_flag;
5         int      f_count;
6         Inode*   f_inode;
7         int      f_offset;
8     };
9

```

其中, `f_flag` 标识位为对打开文件的读、写操作要求, `f_count` 为当前引用该文件控制块的进程数量, `f_inode` 为指向打开文件的内存 Inode 指针, `f_offset` 为文件读写位置指针。

3 详细设计

3.1 重点函数与重点变量

表 2: InodeTable

重点函数与重点变量	功能与意义
<code>Inode m_Inode[NINODE]</code>	内存 Inode 数组, 每个打开文件都会占用一个内存 Inode
<code>Inode* IGet(int inumber)</code>	根据外存 Inode 编号获取对应 Inode
<code>void IPut(Inode* pNode)</code>	减少该内存 Inode 的引用计数
<code>void UpdateInodeTable()</code>	将所有被修改过的内存 Inode 更新到对应外存 Inode 中
<code>int IsLoaded(int inumber)</code>	检查编号为 inumber 的外存 inode 是否有内存拷贝
<code>Inode* GetFreeInode()</code>	在内存 Inode 表中寻找一个空闲的内存 Inode

表 3: OpenFileTable

重点函数与重点变量	功能与意义
<code>FFile m_File[NFILE]</code>	系统打开文件表
<code>File* FAlloc()</code>	在系统打开文件表中分配一个空闲的 File 结构
<code>void CloseF(File* pFile)</code>	对打开文件控制块 File 结构的引用计数 <code>f_count</code> 减 1, 当 <code>f_count</code> 为 0 时释放 File 结构

表 4: OpenFiles

重点函数与重点变量	功能与意义
File* k_OpenFileTable[NOFILES]	File 对象的指针数组, 指向系统打开文件表中的 File 对象
int AllocFreeSlot()	在内核打开文件描述符表中分配一个空闲表项
File* GetF(int fd)	找到对应的打开文件控制块 File 结构
void SetF(int fd, File* pFile)	为已分配到的空闲描述符 fd 和已分配的打开文件表中空闲 File 对象建立勾连关系

表 5: BufferManager

重点函数与重点变量	功能与意义
Buf _bFreeList	自由缓存队列控制块, 定义有自由队列队头
Buf m_Buf[NBUF]	缓存控制块数组
Buf bDevtab	设备控制块, 定义有设备队列队头
char Buffer[NBUF][BUFFER_SIZE]	缓冲区数组
Buf* GetBlk(int blkno)	申请一块缓存, 用于读写字符块 blkno
void Brelse(Buf* bp)	释放缓存控制块 buf
Buf* Bread(int blkno)	读一个磁盘块, blkno 为目标磁盘块逻辑块号
void Bwrite(Buf* bp)	写一个磁盘块
void Bdwrite(Buf* bp)	延迟写磁盘块
void Bawrite(Buf* bp)	异步写磁盘块
void ClrBuf(Buf* bp)	清空缓冲区内容
void Bflush()	将 dev 指定设备队列中延迟写的缓存全部输出到磁盘
void __NotAvail(Buf* bp)	从自由队列中摘下指定的缓存控制块 buf

表 6: FileSystem

重点函数与重点变量	功能与意义
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, 用于创建新的文件
void IFree(int number)	释放编号为 number 的外存 INode, 用于删除文件
Buf* Alloc()	分配空闲磁盘块
void Free(int blkno)	释放编号为 blkno 的磁盘块

表 7: FileManager

重点函数与重点变量	功能与意义
Inode* NameI(char(*func)(), enum DirectorySearchMode mode)	路径搜索
static char NextChar()	获取路径中的下一个字符
void Open()	打开一个文件
void Creat()	新建一个文件
void Close()	关闭一个文件
void ChDir()	改变当前工作目录
Inode* MakNode(unsigned int mode)	新建一个文件时, 分配资源
void Read()	读文件
void Write()	写文件
void Seek()	改变当前读写指针的位置
void Delete()	删除文件
void Open1(Inode* pInode, int mode, int trf)	Open 和 Create 方法的公共部分
void Rdwr(enum File::FileFlags mode)	Read 和 Write 方法的公共部分

表 8: _kernel

重点函数与重点变量	功能与意义
void initialize()	内核初始化
void callInit()	每个函数调用的初始化工作
void format()	格式化磁盘
int open(char* pathname, int mode)	打开文件
int create(char* pathname, int mode)	新建文件
void mkdir(char* pathname)	新建目录
void cd(char* pathname)	改变当前工作目录
void ls()	显示当前目录下的所有文件
int fread(int readFd, char* buf, int nbytes)	读一个文件到目标区
int fwrite(int writeFd, char* buf, int nbytes)	根据目标区的字符写一个文件
void fseek(int seekFd, int offset, int ptrname)	改变读写指针的位置
void fdelete(char* pathname)	删除文件
void fmount(char* from, char* to)	将本机文件拷贝到文件系统磁盘某目录下
int close(int fd)	关闭文件
void clear()	系统关闭时收尾工作

3.2 实现流程

3.2.1 盘块申请

这个算法的目标是从空闲磁盘块中分配一个可用的磁盘块, 并返回该磁盘块的缓冲区, 以便可以在其中进行读取和写入操作。如果没有可用的空闲磁盘块, 将返回 NULL, 并设置对应错误码。

图 1 展示了盘块分配流程图。

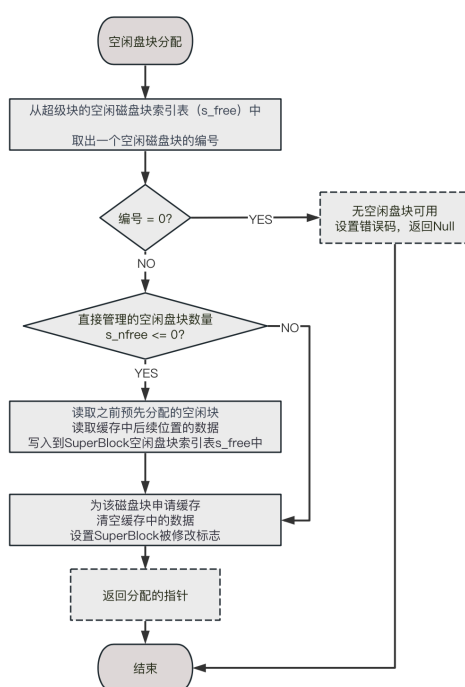


图 1: 盘块分配

3.2.2 盘块释放

这个算法的目标是将释放的磁盘块号添加到超级块的空闲磁盘块索引表中，以便在需要时可以重新分配给其他文件或数据。当空闲磁盘块索引表已满时，将索引表中的数据写入磁盘，并清空索引表，为新的空闲磁盘块腾出空间。

图 2 展示了盘块释放流程图。

3.2.3 DiskInode 申请

该算法的目标是分配空闲的外存 Inode 节点供文件系统使用，并确保分配的 Inode 节点是空闲的。如果没有符合条件的空闲 Inode 可用，将持续循环直到分配成功或者返回 NULL。

图 3 展示了 DiskInode 分配流程图。

3.2.4 DiskInode 释放

该算法用于将不再使用的 Inode 编号添加到空闲 Inode 索引表中，以便后续可以重新分配给新的文件或目录。这样做可以有效地管理和重复利用 Inode 资源。

图 4 展示了 DiskInode 释放流程图。

3.2.5 索引映射

该算法用于将文件的逻辑块号 (lbn) 映射到对应的物理盘块号 (phyBlkno)。对于小型文件，直接从基本索引表中获取物理盘块号；对于大型和巨型文件，需要通过索引表的层次结构进行查找和分配。算法根据文件类型和逻辑块号的范围，按照一级索引、二级索引的方式逐级查找和分配物

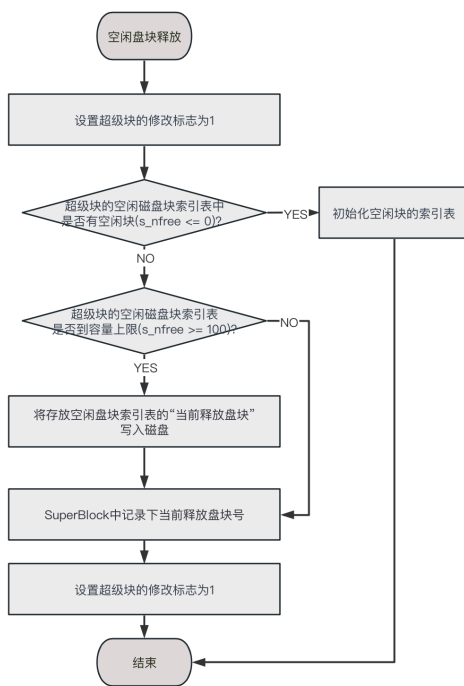


图 2: 盘块释放

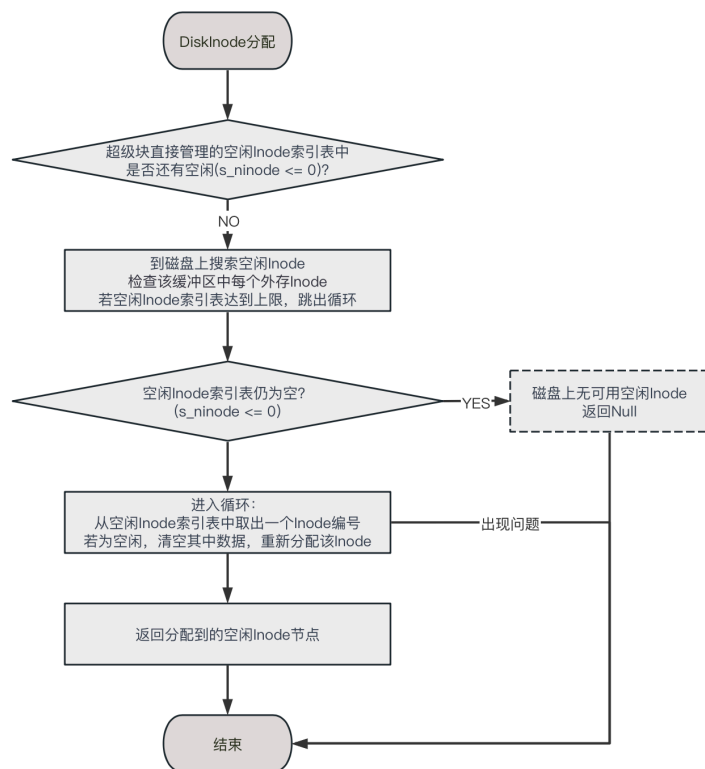


图 3: DiskInode 分配

理盘块号。如果索引表不存在, 则会分配新的物理块, 并将相应的索引信息记录在上一级索引表中。最终, 返回文件的物理盘块号。

这样, 算法确保了文件数据的连续存储和索引的有效管理, 使得文件系统可以根据逻辑块号快

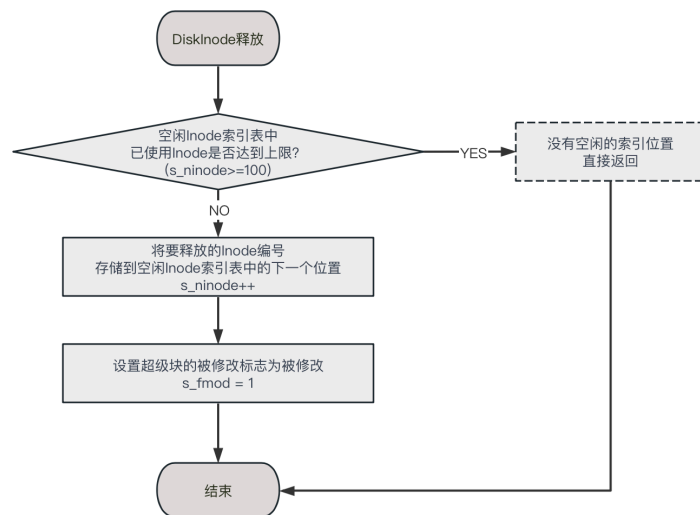


图 4: DiskInode 释放

速定位和读取相应的物理盘块，实现了文件的高效存储和检索。

图 5 展示了盘块映射流程图。

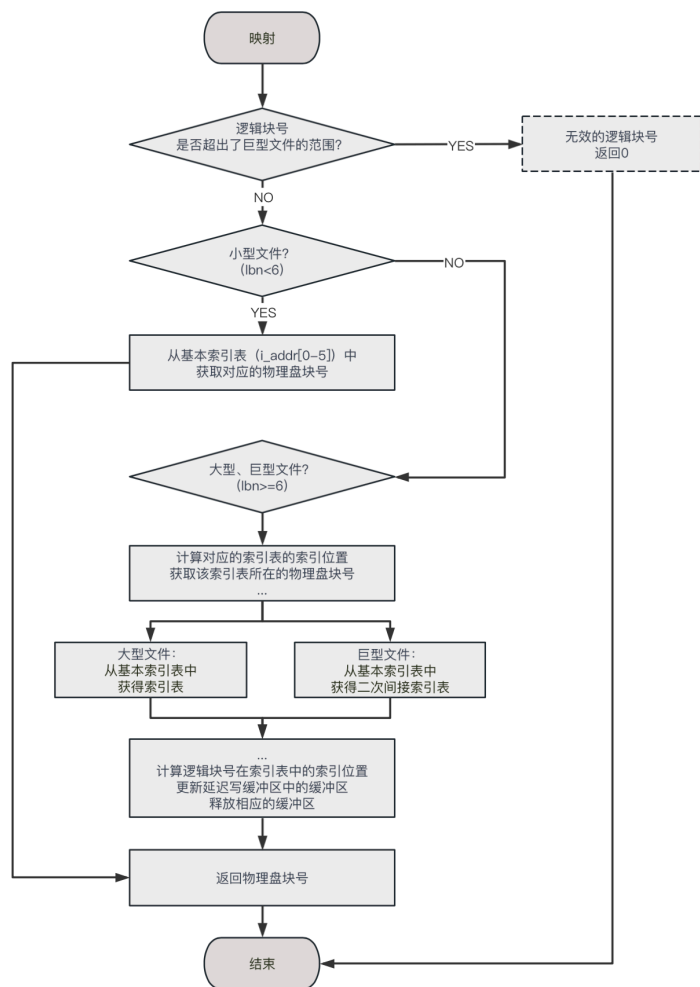


图 5: 映射

3.2.6 目录查找

该算法通过遍历目录的目录项来查找和匹配路径名中的每个路径分量，同时处理特殊情况 and 错误情况，最终返回对应的 Inode 指针或 NULL。这样，算法实现了路径名解析和目录项搜索的功能。

图 6 展示了目录查找流程图。

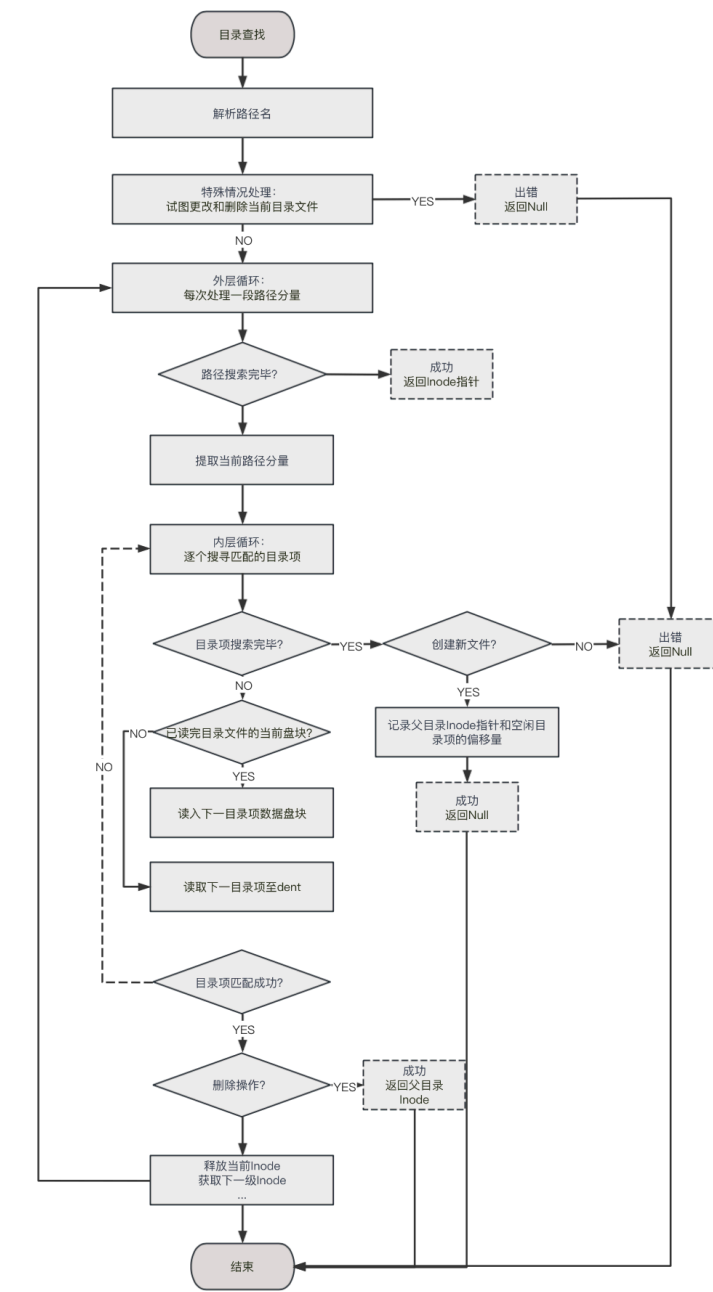


图 6: 目录查找

3.2.7 缓存替换

本项目文件系统中，高速缓存块采取 LRU 算法，即：在需要替换时，选择替换最久未被访问的缓存块进行替换，为新的缓存块腾出空间。

1. 缓存申请

图 7 展示了缓存申请流程图。

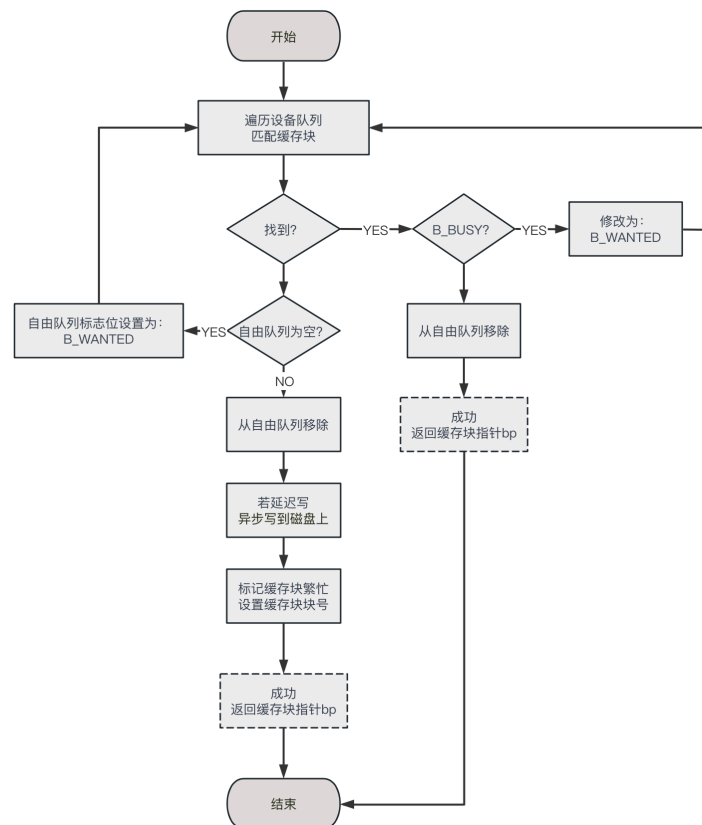


图 7: 缓存申请

2. 缓存释放

图 8 展示了缓存释放流程图。

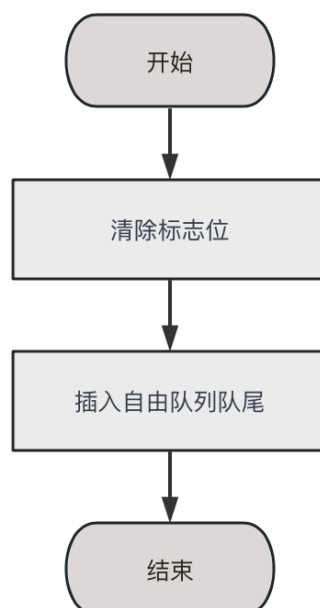


图 8: 缓存释放

3.3 项目结构

项目工程结构如图 9 所示。

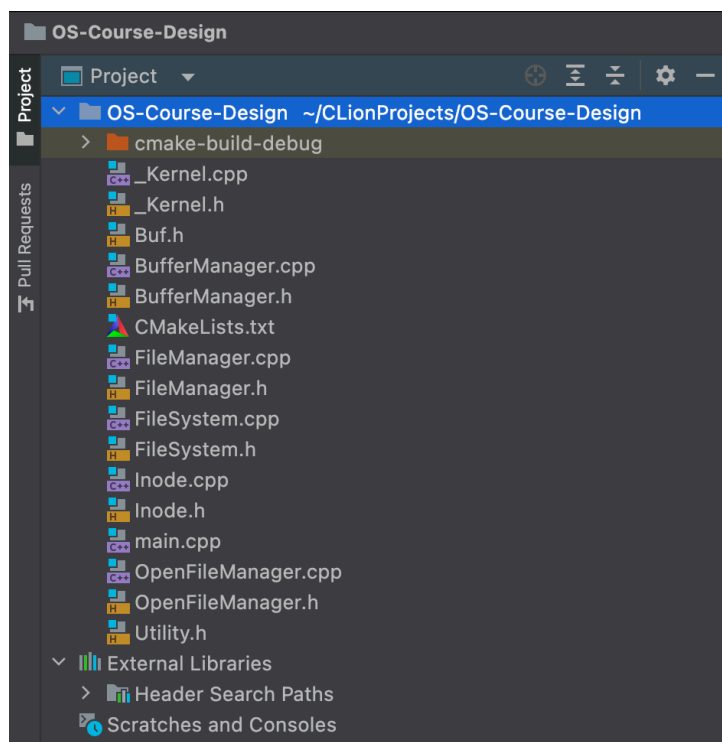


图 9: 项目结构

4 测试与分析

4.1 简单命令测试

在这个部分，对项目支持的指令进行逐条测试与结果分析，具体如下。

1. 初始界面

图 10 展示项目系统的初始化界面。命令行参考 Mac 电脑的 Terminal 输出，其中：siyu 表示电脑用户名，MacBook 表示当前电脑，后续无显示因为处于系统根目录，“%”后等待输入系统指令。

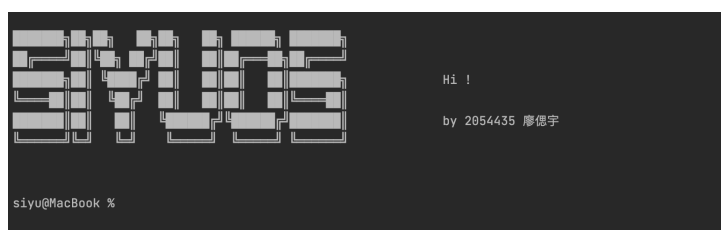


图 10: 初始界面展示

2. fformat

图 11 展示 `fformat` 指令测试，目的为进行格式化。由于此操作变动较大，提示用户再次输入指令以确认进行格式化操作。

```
siyu@MacBook % fformat
This command is used to format disk. Please use with caution!
Enter 'fformat' again to confirm the format operation: fformat
FORMAT SUCCESS!

siyu@MacBook %
```

图 11: 指令测试: `fformat`

3. `mkdir ls`

图 12 展示 `mkdir` 指令测试，目的为创建目录。在图中，完成了根目录下 `home` 目录的创建，键入 `ls` 指令，发现 `home` 目录存在，说明目录创建成功。

```
siyu@MacBook % mkdir home

siyu@MacBook % ls
home
```

图 12: 指令测试: `mkdir ls`

4. `cd`

图 13 展示 `cd` 指令测试，目的为切换路径。在图中，从根目录切换到之前创建的 `home` 子目录下，观察 `%` 前内容可观察到切换成功。另外，也可输入绝对路径完成切换，请参见后续测试内容。

```
siyu@MacBook % cd home

siyu@MacBook home %
```

图 13: 指令测试: `cd`

5. `fcreat`

图 14 展示 `fcreat` 指令测试，目的为创建文件。在图中，创建了 `test.txt` 文本文件，提示创建成功，并返回文件描述符 `0` 以方便用户进行文件操作。同样地，键入 `ls` 指令，发现 `test.txt` 存在，说明文件创建成功。

```
siyu@MacBook home % fcreat test.txt
The file test.txt is created, fd = 0

siyu@MacBook home % ls
test.txt

siyu@MacBook home %
```

图 14: 指令测试: `fcreat`

6. `fclose`

图 15 展示 `fclose` 指令测试，目的为关闭文件。在图中，刚才创建的文件 `test.txt` 的文件描述符为 `0`，输入 `fclose 0` 即完成文件关闭。

```
siyu@MacBook home % cat test 0
siyu@MacBook home %
```

图 15: 指令测试: fclose

7. fopen

图 16 展示 fopen 指令测试, 目的为打开文件。在图中, 重新打开了 test.txt 文本文件, 提示文件打开成功, 分配文件描述符 0。

```
siyu@MacBook home % fopen test.txt
The file test.txt is opened, fd = 0
siyu@MacBook home %
```

图 16: 指令测试: fopen

8. fwrite

图 17 展示 fwrite 指令测试, 目的为对文件进行写操作。在图中, 向 test.txt 文本文件写入了 12 个字节的内容, 提示文件写入成功 (提示实际写入字节数)。

```
siyu@MacBook home % fwrite 0 12 Hello,world!
fwrite: Actually write 12 bytes.
siyu@MacBook home %
```

图 17: 指令测试: fwrite

9. fread

图 18 展示 fread 指令测试, 目的为对文件进行读操作。在图中, 读取刚才已写入内容的 test.txt 文本文件, 并依次读取了 5 个字节和 3 个字节。对照之前写入内容可知, 文件读操作命令正确。

```
siyu@MacBook home % fread 0 5
Hello
fread: Actually read 5 bytes.

siyu@MacBook home % fread 0 3
,wo
fread: Actually read 3 bytes.

siyu@MacBook home %
```

图 18: 指令测试: fread

10. fseek

图 19 展示 fseek 指令测试, 目的为定位文件读写指针。在图中, 重新打开 test.txt 文本文件 (此时文件指针置于开头), 并键入 fseek 0 6 0 命令以将指针置于第 6 个字节后, 然后再次进行读操作, 对照之前写入内容可知, 文件读写指针定位成功。

11. fdelete

图 20 展示 fdelete 指令测试, 目的为删除文件。在图中, 删除了 test.txt 文本文件, 结合删除操作前后两次 ls 指令结果可知, 文件删除成功。

```
siyu@MacBook home % open test.txt
The file test.txt is opened, fd = 0

siyu@MacBook home % fseek 0 0

siyu@MacBook home % fread 0 0
world
fread: Actually read 5 bytes.

siyu@MacBook home %
```

图 19: 指令测试: fseek

```
siyu@MacBook home % rm
test.txt

siyu@MacBook home % rm test.txt

siyu@MacBook home %

siyu@MacBook home %
```

图 20: 指令测试: fdelete

12. exit

图 21 展示 exit 指令测试，目的为登出文件系统。在图中，参考 Mac 终端 Terminal 的退出提示符，打印文件系统退出信息。

```
siyu@MacBook % exit

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

Goodbye! Have a great day!

[Process completed]

Process finished with exit code 0
|
```

图 21: 指令测试: exit

4.2 复杂命令测试 · 存放指定文件

在这个部分，对课程设计要求的命令行操作进行测试与结果分析，首先给出课设要求：

通过命令行方式完成下列操作：

- 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；
- 把你的课设报告，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。

下面开始测试：

1. 格式化文卷

图 22 展示磁盘问卷格式化操作，即之前的测试内容全被清除。

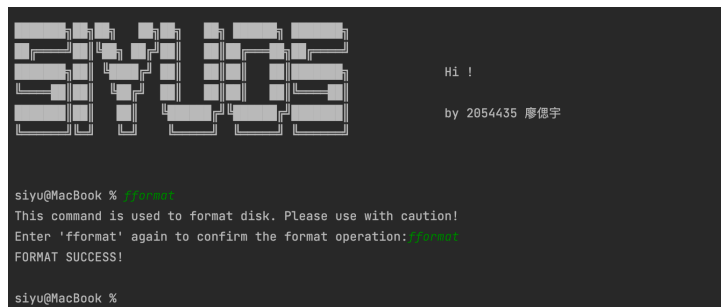
A terminal window on a Mac. At the top left, the word 'HUGO' is displayed in a large, stylized font made of small squares. To the right, it says 'Hi !' and 'by 2054435 廖偲宇'. The terminal shows the command 'siyu@MacBook % fformat' being entered. Below it, a message says 'This command is used to format disk. Please use with caution! Enter 'fformat' again to confirm the format operation:'. The user enters 'fformat' again, and the terminal responds with 'FORMAT SUCCESS!'. The prompt returns to 'siyu@MacBook %'.

图 22: 复制测试 1: 格式化文卷操作

2. 一级目录创建

图 23 展示在根目录下创建一级目录，即 bin、etc、home、dev。目录创建通过 mkdir 命令完成，创建完成后键入 ls，由结果可知，四个子目录创建成功。

A terminal window showing the creation of primary directories. The user enters 'siyu@MacBook % mkdir bin', 'siyu@MacBook % mkdir etc', 'siyu@MacBook % mkdir home', and 'siyu@MacBook % mkdir dev' in sequence. Finally, the user enters 'siyu@MacBook % ls', and the terminal outputs 'bin etc home dev'.

图 23: 复杂测试 1: 一级目录创建

3. 二级目录创建

图 24 展示在 home 目录下创建二级目录，即 texts、reports、photos。目录创建通过 mkdir 命令完成，由结果可知，三个子目录创建成功。

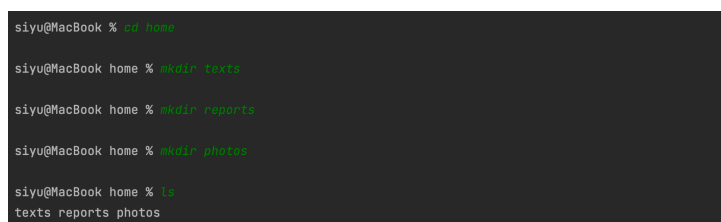
A terminal window showing the creation of secondary directories. The user first enters 'siyu@MacBook % cd home' to navigate to the home directory. Then, they enter 'siyu@MacBook home % mkdir texts', 'siyu@MacBook home % mkdir reports', and 'siyu@MacBook home % mkdir photos' in sequence. Finally, the user enters 'siyu@MacBook home % ls', and the terminal outputs 'texts reports photos'.

图 24: 复杂测试 1: 二级目录创建

4. 文件创建

图 25 展示在 home 目录下对应的三个子目录里（texts、reports、photos）创建要求的三个文件的过程。其中，路径的切换以绝对路径方式完成。

5. 文件装载

图 26 展示向创建的三个文件装载实际文件的过程。具体的方式为实现了一个 fmount 指令，完成本机文件向文件系统写入的操作。

```

siyu@MacBook home % cd texts

siyu@MacBook home/texts % ./create ReadMe.txt
The file ReadMe.txt is created, fd = 0

siyu@MacBook home/texts % ./close 0

siyu@MacBook home/texts % cd /home/reports

siyu@MacBook home/reports % ./create report.pdf
The file report.pdf is created, fd = 0

siyu@MacBook home/reports % ./close 0

siyu@MacBook home/reports % cd /home/photos

siyu@MacBook home/photos % ./create photo.png
The file photo.png is created, fd = 0

siyu@MacBook home/photos % ./close
fclose: missing operand
Usage: fclose [fd]

siyu@MacBook home/photos % ./close 0

```

图 25: 复杂测试 1: 文件创建

```

siyu@MacBook home/photos % cd /home/texts

siyu@MacBook home/texts % ls
ReadMe.txt

siyu@MacBook home/texts % ./mount /Users/lapdeng/test/test/ReadMe.txt ReadMe.txt

siyu@MacBook home/texts % cd /home/photos

siyu@MacBook home/photos % ls
photo.png

siyu@MacBook home/photos % ./mount /Users/lapdeng/test/test/photo.png photo.png

siyu@MacBook home/photos % cd /home/reports

siyu@MacBook home/reports % ls
report.pdf

siyu@MacBook home/reports % ./mount /Users/lapdeng/test/test/report.pdf report.pdf

siyu@MacBook home/reports %

```

图 26: 复杂测试 1: 文件装载

4.3 复杂命令测试 · 文件写入定位

在这个部分，对课程设计要求命令行操作进行测试与结果分析，首先给出课设要求：

通过命令行方式完成下列操作：

- 新建文件/test/Jerry，打开该文件，任意写入 800 个字节；
- 将文件读写指针定位到第 500 字节，读出 500 个字节到字符串 abc；
- 将 abc 写回文件。

下面开始测试：

1. 目录创建

图 27 展示在根目录下再次新建 test 目录。通过两次 ls 命令可知，test 目录创建成功。

2. 文件创建

```
siyu@MacBook % ls
bin etc home dev

siyu@MacBook % mkdir test

siyu@MacBook % ls
bin etc home dev test
```

图 27: 复杂测试 2: 目录创建

图 28 展示在 test 目录下创建测试所需 Jerry 文件的过程, 文件创建完成后系统提示反馈分配给 Jerry 文件的文件描述符为 0。通过 ls 命令可知, Jerry 文件创建成功。

```
siyu@MacBook % cd test

siyu@MacBook test % touch Jerry
The file Jerry is created, fd = 0

siyu@MacBook test % ls
Jerry
```

图 28: 复杂测试 2: 文件创建

3. 写入 800 字节

图 29 展示向 Jerry 文件写入 800 字节的过程。为了方便, 事先通过 python 脚本生成了 800 字节的随机字符保存到本机的 800_bytes.txt 文件中, 然后通过 fmount 命令装载至 Jerry 文件。

```
siyu@MacBook test % fmount /Users/siyu/Desktop/800_bytes.txt Jerry
```

图 29: 复杂测试 2: 写入 800 字节

4. 读出 500 字节

图 30 展示由 Jerry 文件 500 字节处开始读出 500 字节的过程。首先通过 fseek 命令完成定位, 采用 fread 命令开始读出。由于文件只有 800 字节, 提示实际读出了 300 字节。经同源文件对比得知, 文件读出正确。

```
siyu@MacBook test % fseek 500
siyu@MacBook test % fread 500
fread: Actually read 300 bytes.
```

图 30: 复杂测试 2: 读出 500 字节

5. 写回文件

图 31 展示由 Jerry 文件写回读出内容的过程。具体的方式为通过 fwrite 指令, 写回刚才读出的字符串, 系统反馈实际写入 300 字节。测试全部完成, 关闭 Jerry 文件。

```
siyu@MacBook test % fwrite 300
fwrite: Actually write 300 bytes.

siyu@MacBook test %
```

图 31: 复杂测试 2: 写回文件

5 课设总结

本次操作系统课设中，我完成了一个类 Unix 文件系统的编写，该文件系统实现了文件和目录的创建、读写等基本操作。我采用了 C++ 语言作为主要的编程语言，并在 MacOS (CLion) 环境下进行了开发和测试。

在开发过程中，我也遇到了一些挑战和困难，例如：如何正确地设计代码架构，方便项目的编写；怎么正确处理磁盘空间的分配和释放、设计各种数据结构等等。通过尽可能充分理解文件系统的原理和相关概念，并参考了相关文献和资料，同现有的系统进行对比，我逐步解决了这些问题，并完成了文件系统的实现。

通过完成本次课设，我对操作系统的文件系统有了更深入的理解，并在实践中逐渐熟悉了文件和管理、磁盘空间的分配和释放等核心概念和技术。这次课设提高了我的编程能力和解决问题的能力，并让我对操作系统的原理和设计有了更全面的认识。

总之，通过本次课设，我不仅完成了一个类 Unix 文件系统的编写，还提升了对操作系统的理解和实践能力。虽然过程略有些痛苦，但收获颇丰。

6 参考资料

本项目完成过程中，参考资料如下：

1. Unix V6++ 源码
2. 大三·上 (2022-2023) 课程资料
3. GitHub 上林佳奕同学的课程报告；吴子睿同学的代码架构（具体实现不同）