# 第 4 章 文件系统

## 源码阅读

**阅读要求：**

文件系统部分 buf.h fcntl.h stat.h fs.h file.h ide.c bio.c log.c fs.c file.c sysfile.c exec.c

**其他文件系统参考：**

由于本章存在大量的源码需要配合问题来解释。因而，这里不详述。这里介绍下，Unix和Window的文件系统：

Unix文件系统是以块(Block)为单位对磁盘进行读写的。一般而言，一个块的大小为512Byte或者4KB。文件系统的所有数据结构都以块为单位存储在硬盘上，一些典型的数据块包括：超级数据块, i节点块, 数据块。

Window的文件系统FAT磁盘物理上分为四部分组成：保留区(含MBR–Main Boot Record)、保留区(含DBR–DOS Boot Record)、FAT区、数据区。保留区含有一个重要的数据结构–系统引导扇区DBR。FAT12、FAT16的保留区通常只有一个扇区。而FAT32保留的多些，除了0号扇区外，还有一些其他的扇区，其中包括了DBR的备份扇区。

FAT32中簇地址是用4字节进行编址的，故在FAT表中，是以4个字节为单位进行划分，每个单元存储一个簇地址。 \*\*0号地址与1号地址被系统保留并存储特殊标志内容。\*\*从2号地址开始，第i号地址对应数据区中i号簇。我们称 FAT表中的地址为FAT表项，FAT表中记录的值为FAT表项值。

当文件系统被创建时，FAT表会被清空，在FAT1和FAT2表中的0号地址与1号地址会被写入特定值。由于创建文件系统的同时会创建根目录，也就是在数据区为根目录分配一个簇的空间(2号簇，起始簇)在FAT表中2号地址写入一个结束标记。当FAT表中第i号地址对应的簇未被使用时，表项值为0

## 讨论总结

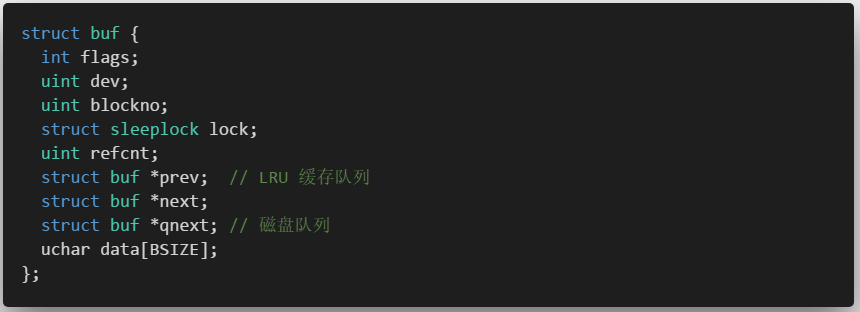
**了解 UNIX 文件系统的主要组成部分：超级块（superblock）， i 节点（inode）， 数据块 （data block），目录块（directory block），间接块（indirection block）**

由于上文介绍了Unix文件系统。这里阐述一下各个部分的功能：

1. 超级块：相当于是保存了文件系统的元数据信息，描述文件系统的状态，比如它有多大，何处有空闲空间以及其他一些信息等等。
2. i节点：i节点表相当于是i节点的一个数组，内核所引用的i节点号即是i节点在该数组中的下标大小。类似于文件的元数据信息。
3. 数据块：真正存放数据的地方。
4. 目录块：在数据块中保存的数据是一系列的i节点 <-> 文件名的 item。
5. 间接块：i节点中包含了几个数据块的地址。此外，当文件过大，10个数据块无法存放完毕，系统会动态分配空间用于存储这些超出信息存放的数据块地址，这就是间接块。I节点后三位存放的是间接块。

**阅读文件 ide.c**

在ide.c文件中，XV6实现了一个简单的基于Programmed IO (非DMA)的IDE驱动程序代码。一个Xv6中的磁盘读写请求的数据结构如下：



在IDE中，还实现了下列方法，其具体功能如下：

| **函数名** | **功能** |
| --- | --- |
| idewait() | 等待磁盘进入空闲状态 |
| ideinit() | 初始化IDE磁盘IO |
| idestart() | 开始一个磁盘读写请求 |
| iderw() | 上层文件系统调用的磁盘IO接口 |
| ideintr() | 当磁盘请求完成后中断处理程序会调用的函数 |

当系统启动时，main()主函数会通过ideinit()方法来对IDE磁盘进行初始化操作。

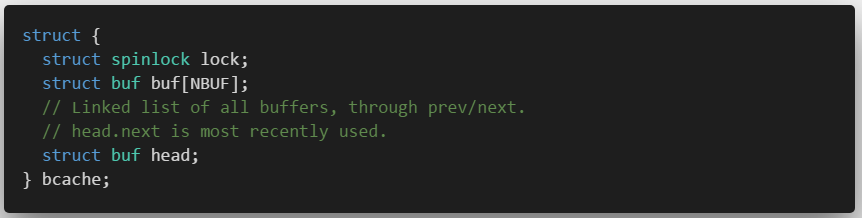
而iderw()方法的主要功能是：提供面向顶层文件系统模块的接口。iderw()方法可用于磁盘的读写操作，需要通过检查buf->flag里的DIRTY位和VALID位来判断是读操作还是写操作。如果请求队列为空，则说明当前磁盘没有处在工作状态，则需要调用idestart()方法初始化磁盘请求队列，并设置中断。当执行完以上操作，iderw()会将调用者切换到睡眠状态。

当磁盘读取或者写操作完毕时，会触发中断进入trap.c中的trap()方法，trap()方法会调用ideintr()方法处理磁盘相关的中断。在ideintr()方法中，如果当前请求是读请求，就读取目前已经在磁盘缓冲区中准备好的数据。最后，ideintr()方法会唤醒正在睡眠等待当前请求的进程，如果队列里还有请求，就调用idestart()方法来处理新的请求。

**了解 XV6 文件系统中 buffer cache 层的内容和实现。描述 buffer 双链表数据结构及其初始化过程。了解 buffer 的状态。了解对 buffer 的各种操作。**

在文件系统中，Buffer Cache担任了一个磁盘与内存文件系统交互的中间层。由于对磁盘的读取是非常缓慢的，因此将最近经常访问的磁盘块缓存在内存里是很有益处的。

Xv6中Buffer Cache的实现在bio.c中，Buffer Cache的数据结构如下：



其中，最重要的数据是在固定长度的数组上维护了一个由buf组成的双向链表。此外，lock保护对Buffer Cache链表结构的访问。值得注意的是，对链表结构的访问和对一个buf结构的访问需要的是不同的锁。

在系统对缓存进行初始化时，系统调用binit()方法。binit()方法对缓存内每个元素初始化睡眠锁，并从后往前连接成一个双向链表。

操作系统使用bread()方法和bwrite()方法对缓存中的块进行读写操作。关于缓存的全部操作是在bread()方法与bwrite()方法中自动完成的，不需要上层文件系统的参与。

bread()方法：首先，调用bget()方法，而bget()方法会检查请求的磁盘块是否在缓存中。如果磁盘块在缓存中，则直接返回缓存中对应的磁盘块即可。如果不在缓存中，那么需要先使用最底层的iderw()函数先将此磁盘块从磁盘加载进缓存中，再返回此磁盘块。

bwrite()函数的作用是:将缓存中的数据直接写入磁盘。Buffer Cache层不尝试执行任何延迟写操作，对磁盘的bwrite()写操作的时间由上层文件系统控制。

上面的文件系统调用brelse()方法来释放不再使用的脉冲块。brelse()方法主要处理双向链表上的操作，这里不再重复。

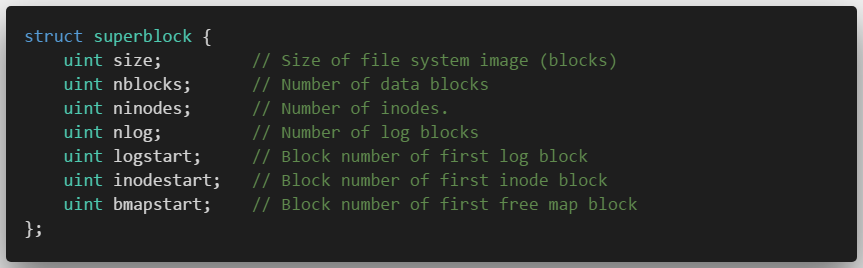
**了解 XV6 文件系统中的 logging 和 transaction 机制。**

在文件系统中添加Log层是为了能够使文件系统能够处理电源系统等异常，避免磁盘文件系统上的不一致。Log层的实现思路是，对于文件系统顶部全部是磁盘操作，它分为一个个transaction。首先，将所有数据及其对应的每个事务Log区域上的磁盘号写入磁盘，并且只有在写入Log区域后才完成，然后 数据被写入日志区域实际数据存储区域。通过此设计，如果禁用对日志的写入，则文件系统将假定写入不存在，并且如果禁用对实际区域的写入，则Log区域数据可用于恢复文件系统。这样，可以避免文件系统中文件的损坏。

使用Log层时，上层文件系统必须首先调用begin\_op（）方法。 begin\_op（）方法记录一个新事务。 使用日志层后，上层系统必须调用end\_op（）方法。 仅当没有事务正在执行时，日志才会执行实际磁盘写入。 在commit（）方法中真正的磁盘写操作，你可以看到只在end\_op（）结束时调用commit（）方法，log.outstanding == 0（和启动时间）。 commit（）方法首先调用write\_log（）方法将高速缓存中的磁盘块写入磁盘上的Log区域，并将Log Header写入磁盘区域。 仅当更新磁盘上存在日志标题的区域数据时，才会完成日志更新。 更新日志区域后，commit（）方法调用install\_trans（）以完成实际磁盘写入步骤，之后调用write\_head（）方法以清除当前Log数据。

**了解 XV6 文件系统的硬盘布局。**

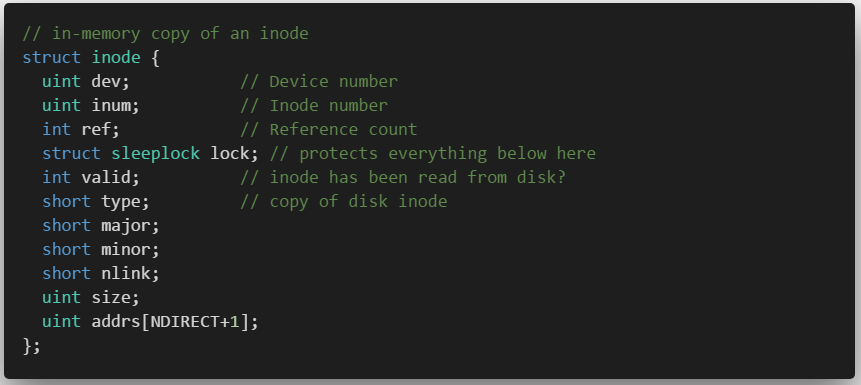
在Xv6操作系统的硬盘中，按顺序存储了几个硬盘块：boot block 、 super block 、 log 、 inode blocks 、 free bit map 、 data blocks。 这些硬盘块的索引直接由整数使用。 第一个硬盘块引导块在引导时加载到内存中，磁盘块号为0.第二个超级块占用硬盘块，编号为1，Xv6中的声明如下：



Superblock中存储有关文件系统的元信息。 操作系统必须首先读取Superblock以了解剩余log块，inode块，bitmap块和data块的大小和位置。 在Superblock之后顺序存储多个日志块，多个inode块和多个位图块。 磁盘的剩余部分存储数据块块。

**了解 XV6 的“文件”有哪些，以及文件，i 节点，设备相关的数据结构。了解 XV6 对文件的基本操作有哪些。XV6 最多支持多少个文件？每个进程最多能打开多少个文件？**

Xv6中的文件（包括目录）都由inode数据结构表示，所有文件的inode都存储在磁盘上。当系统或进程需要使用inode时，inode将被加载到inode缓存中。存储在内存中的inode将具有比存储在磁盘上的inode更多的运行时信息。内存中的inode数据结构声明如下。



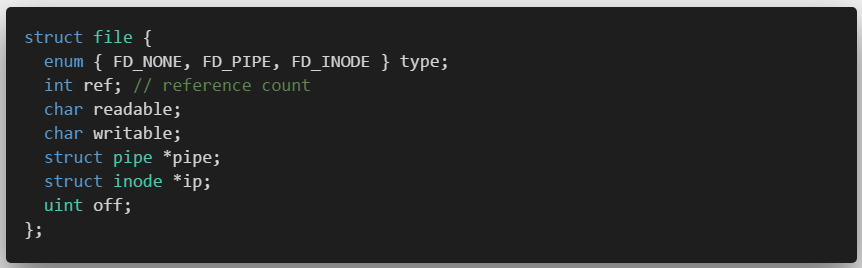
其中，inode.type指明了这个文件的类型。Xv6中，这个类型可以是普通文件，目录，或者是特殊文件。

内核在内存中维护一个i节点缓存。同时，文件中还实现了有关i节点的相关操作。其相关方法及其功能如下：

1. iinit()方法 主要功能是：读取Superblock，初始化inode相关的锁
2. readi()方法 主要功能是：往inode读数据
3. ialloc()方法 主要功能是：在磁盘上分配一个inode
4. ilock()方法 主要功能是：获取指定inode的锁
5. iunlock()方法 主要功能是：释放指定inode的锁
6. iget()方法 主要功能是：获取指定inode，更新缓存
7. writei()方法 主要功能是：往inode写数据
8. bmap()方法 主要功能是：返回inode的第n个数据块的磁盘地址
9. iupdate()方法 主要功能是：将内存里的一个inode写入磁盘
10. iput()方法 主要功能是：对内存内一个Inode引用减1，引用为0则释放inode

Inode具有12个（NDIRECT）直接映射的磁盘块和128个间接映射的磁盘块。 总之，XV6系统支持的最大文件大小为140 \* 512B = 70KB。

在XV6操作系统中，一个文件的数据结构表示如下

由此可见，file数据结构可以表示inode或管道。 多个file数据结构可以抽象相同的inode，但Offset可以是不同的。系统所有的打开文件都在全局文件描述符表ftable中

从中可以看出Xv6最多支持同时打开100(NFILE)个文件，从struct proc中可以看出Xv6中每个进程最多同时可以打开16(NOFILE)个文件。

**了解与文件系统相关的系统调用，简述各个系统调用的作用。**

利用上一层的实现，大多数系统调用的实现都是比较直接的。Xv6中支持的文件相关系统调用如下：

1. sys\_link()方法 主要功能是：为已有的inode创建一个新的名字
2. sys\_unlink()方法 主要功能是：为已有的inode移除一个名字，可能会移除这个inode
3. sys\_open()方法 主要功能是：打开一个指定的文件描述符
4. sys\_mkdir()方法 主要功能是：创建一个新目录
5. sys\_mknod()方法 主要功能是：创建一个新文件
6. sys\_chdir()方法 主要功能是：改变进程当前目录
7. sys\_fstat()方法 主要功能是：改变文件统计信息
8. sys\_read()方法 主要功能是：读文件描述符
9. sys\_write()方法 主要功能是：写文件描述符
10. sys\_dup()方法 主要功能是：增加文件描述符的引用