|  |
| --- |
|  |
| 文件系统阅读报告 |
|  |

|  |
| --- |
| **姓名: 余晶 学号：S1748036**  **姓名: 方明 学号：S1748007**  **日期: 2018.04.05** |

[内容一：任务完成情况 3](#_Toc508263543)

[任务完成列表（Y/N） 3](#_Toc508263544)

[具体Exercise的完成情况 3](#_Toc508263545)

[内容二：参考文献 10](#_Toc508263546)

## 内容一：任务完成情况

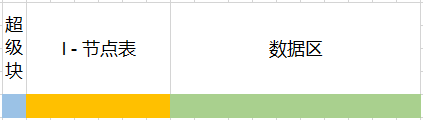
### 任务完成列表（Y/N）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 完成情况 | Y | Y | Y | Y | Y | N | N |

### 具体Exercise的完成情况

1. 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）。分别解释它们的作用。（以下各题内容按照程序的调用关系自底向上编排）

* 超级块：文件系统中的第一块称为超级块。这个块存放文件系统本身的结构信息。也存放未被使用的磁盘块信息；
* i节点：文件系统的下一个部分被称为i节点表。每一个文件都有一些属性，如大小、文件所有者和最近修改时间等。这些信息都存放在i节点的结构中；
* 数据块：文件系统的第三个部分是数据区。文件的内容保存在这个区域。磁盘上所有的块的大小都是一样的。如果文件包含了超过一个块的内容，则文件内容会存放在多个磁盘块中；



* 目录块：是目录项的集合.每一个目录项中包含2个数据:文件名,i节点号.文件名就是通常我们用ls命令所看到的,i节点号就是该目录项所连接到的i节点；
* 间接块：文件内容的分配情况是必须记录在i-节点的磁盘序号列表里的。由于i-节点有一个上限，如果文件块超过了这个上限，把其中一块存放其它块的编号，即这块记录着其它块编号的指针，通过这个指针就能找到它们，这个用来存放编号的数据块就称为间接块；

1. 阅读文件ide.c。这是一个简单的ide硬盘驱动程序，对其内容作大致了解

* 通过阅读代码知，XV6 采用数据结构buf来对磁盘块管理：

struct buf {

int flags;

uint dev;

uint sector;

struct buf \*prev; // LRU cache list

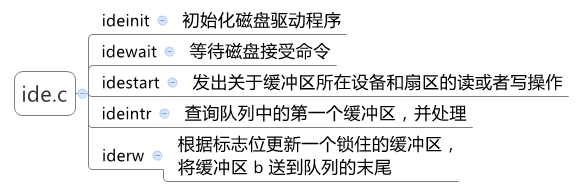
struct buf \*next;

struct buf \*qnext; // disk queue

uchar data[512];

};

通过dev和sector就可以唯一确定该block的地址。读写的粒度为block(512字节)，相应的数据保存在data中。每个buffer含有一个标志flags，如果为B\_BUSY说明这个buffer正在被一个进程使用，如果B\_VALID则说明buffer中的数据已经准备好了，B\_DIRTY说明是读取磁盘数据还是将数据写入磁盘；



* XV6 维持了一个磁盘队列struct buf \*idequeue。每次新增的buffer放在idequeue 尾部。如果队列为空，则调用idestart 处理数据；
* XV6 磁盘中断处理程序调用 ideintr 来处理。它通过访问队列idequeue buffer的flags绝对如何处理数据。如果是读取操作，则将数据读到data中。现在buffer中的数据已经准备好了，则将flags设置为B\_VALID并清除B\_DIRTY,调用wakeup唤醒因为这个buffer而进入睡眠状态的进程。如果队列非空，则调用idestart处理下一个buf；

1. 阅读文件buf.h，bio.c。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。

* buf.h 主要定义了数据结构buf（如上题）和flags的几种状态。磁盘驱动程序用结构体 buf 来表示一个磁盘扇区。每一个缓冲区表示磁盘设备上的一个扇区。dev 和 sector 给出了设备号和扇区号，data 是该磁盘扇区数据的内存中的拷贝。flags 记录了内存和磁盘的联系：B\_VALID 位代表数据已经被读入，B\_DIRTY 位代表数据需要被写出。B\_BUSY 位是一个锁；它代表某个进程正在使用这个缓冲区，其他进程必须等待。当一个缓冲区的 B\_BUSY 位被设置，我们称这个缓冲区被锁住。XV6 以块基本单位对磁盘进行管理，每块的大小为512字节。并采用LRU算法来实现缓存替换；

struct {

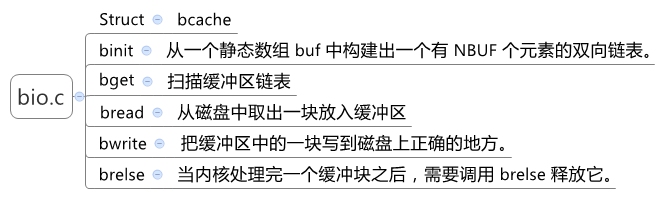
struct spinlock lock;

struct buf buf[NBUF];

struct buf head;

} bcache;

* XV6 通过buf结构体head来构成双向链表，该链表维护了块的使用频率，head.next指示最近使用的块。bio.c定义了：binit、bget、bread、bwrite、brelse方法；



. binit初始化bcache双向链表并设置块缓冲区需要使用的锁；

. bget函数有两个参数：设备号和块编号。bread调用bget得到块缓冲结构。bget扫描缓冲区链表，在块缓冲区中找到缓冲块，如果此缓冲块已经有进程占用，则睡眠当前进程等待唤醒。如果bget没有找到相应的块缓冲结构，则回收一些非繁忙和清洁的缓冲区；

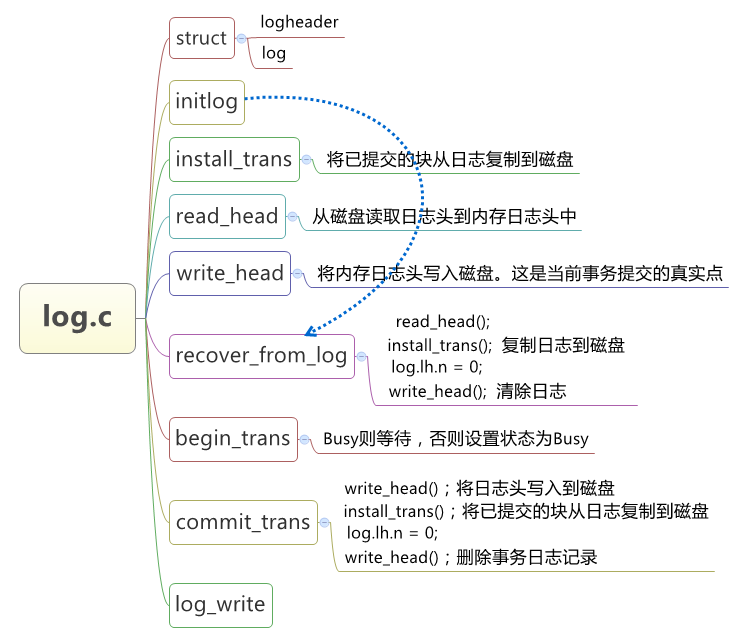
. bread从磁盘中取出一块放入缓冲区；

. bwrite将块缓冲结构写入磁盘；

. brelse释放锁定缓冲区。当调用者使用完了一块缓冲区，他必须调用 brelse 来释放它，brelse将一块缓冲区移动到链表的头部，清除 B\_BUSY，唤醒睡眠在这块缓冲区上的进程；

1. 阅读文件log.c。了解XV6文件系统中的logging和transaction机制。

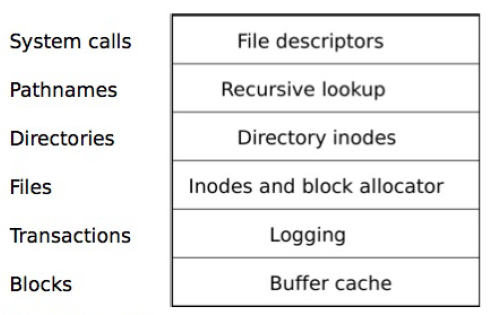
* log.c 的内容如下图。一共包括两个结构体：logheader和log。方法：initlog、install\_trans、read\_head、write\_head、recover\_from\_log、begin\_trans、 commit\_trans和log\_write；



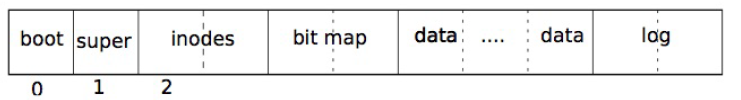
* logheader 包含了block数组用于记录应该写入磁盘的块号，参数n用于记录有效块号；
* 当修改了内存中的块缓冲区后，log\_write 在logheader中记录要写入磁盘的块号。当调用commit\_trans时才真正写入磁盘。commit\_trans 包含三个步骤：write\_head、install\_trans和write\_head。write\_head 将内存中保存的logheader写入磁盘，这是当前事务提交的真实点。然后调用install\_trans将已经提交的块从日志复制到磁盘。最后commit\_trans 会把日志起始块中的计数改为0，这样在下次会话之前的系统崩溃就会使得恢复代码忽略日志；
* begin\_trans 会一直等到它独占了日志的使用权后返回；
* recover\_from\_log 在 initlog 中被调用，而 initlog 在第一个用户进程开始前的引导过程中被调用。它读取日志的起始块，如果起始块说日志中有一个提交了的会话，它就会仿照 commit\_trans 的行为执行，从而从错误中恢复；

1. 阅读文件fs.h，fs.c。了解XV6文件系统的硬盘布局。

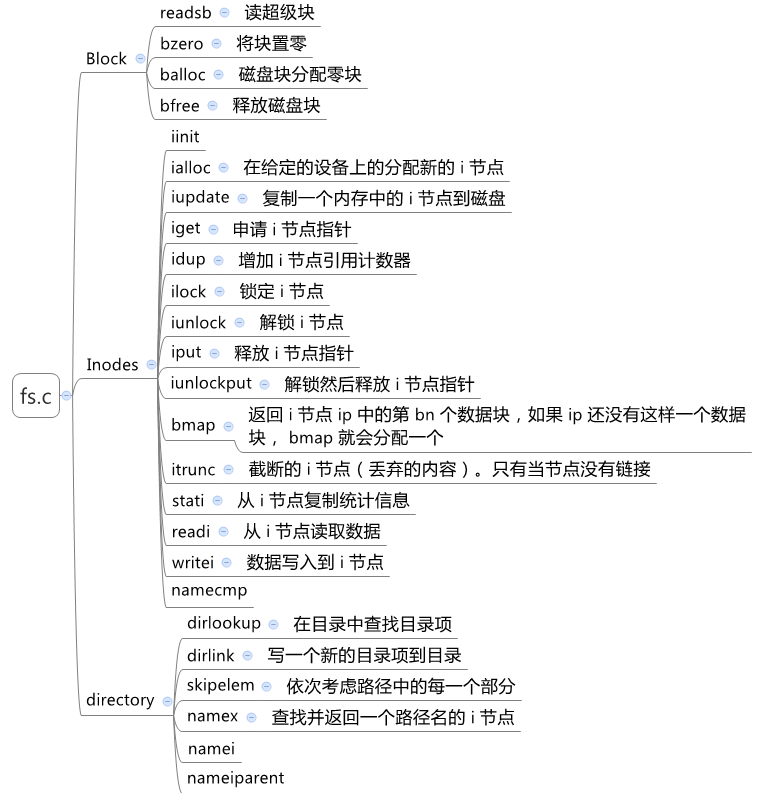
* XV6 的文件系统分6层实现，如下图所示。最下面一层通过块缓冲读写 IDE 硬盘，它同步了对磁盘的访问，保证同时只有一个内核进程可以修改磁盘块。第二层使得更高层的接口可以将对磁盘的更新按会话打包，通过会话的方式来保证这些操作是原子操作（要么都被应用，要么都不被应用）。第三层提供无名文件，每一个这样的文件由一个 i 节点和一连串的数据块组成。第四层将目录实现为一种特殊的 i 节点，它的内容是一连串的目录项，每一个目录项包含一个文件名和对应的 i 节点。第五层提供了层次路经名，这一层通过递归的方式来查询路径对应的文件。最后一层将许多UNIX 的资源（如管道，设备，文件等）抽象为文件系统的接口，极大地简化了程序员的工作［4］；



* 文件系统必须设计好在磁盘上的什么地方放置 i 节点和数据块。XV6 把磁盘划分为几个区块，如下图所示。文件系统不使用第0块（第0块存有 bootloader）。第1块叫做超级块；它包含了文件系统的元信息（如文件系统的总块数，数据块块数，i 节点数，以及日志的块数）。从第2块开始存放 i 节点，每一块能够存放多个 i 节点。接下来的块存放空闲块位图。剩下的大部分块是数据块，它们保存了文件和目录的内容。在磁盘的最后是日志块，它们是会话层的一部分［4］；



* 在fs.h 中定义了超级块的数据结构superblock和i 节点的数据结构 dinode；
* 在fs.c中定义了三类方法：块、i 节点和文件目录相关的方法，如下图；



* 块分配器提供了两个功能：balloc 分配一个新的磁盘块，bfree 释放一个块。balloc调用 readsb 从磁盘中读取超级块到 sb 中。 然后通过两层循环找到一个零块；
* 要申请一个新的i 节点, XV6 会调用 ialloc方法，分配一个i 节点。ialloc 逐块遍历i 节点表，寻找一个 type 为0的空闲i 节点。然后修改 type，调用 iget 返回。iget 遍历i 节点缓存寻找一个给定的设备和i 节点号的项，如果找到一项，就返回对这个i 节点的引用。调用者在读写 i 节点的元数据或内容之前必须用 ilock 锁住 i 节点。 ilock （4703）用一个类似的睡眠循环来等待 ip->flag 的 I\_BUSY 位被清除，而后由自己再设置它。一旦 ilock 拥有了对 i 节点的独占，他可以根据需要从磁盘中读取出 i 节点的元数据。函数 iunlock 用来清除 I\_BUSY 位并且唤醒睡眠在 ilock 中的其他进程。iput 用于释放指向i 节点的指针，将引用计数器值减1。当减到0，那么这个i 节点就会变为空闲状态，可以被另外一个i 节点重用。如果 iput 发现没有指针指向一个 i 节点并且也没有任何目录项指向它，那么这个 i 节点和它关联的数据块都应该被释放。 iput 重新锁上这个 i 节点，调用 itrunc 来把文件截断为0字节，释放掉数据块；把 i 节点的类型设置为0；把变化写到磁盘中；最后解锁 i 节点。bmap 返回 i 节点 ip 中的第 bn 个数据块，如果 ip 还没有这样一个数据块， bmap 就会分配一个。readi 从 i 节点中读出数据。writei 将数据写入 i 节点；
* 目录的 i 节点的类型 T\_DIR , 它的数据是一系列的目录条目。每个条目是一个 struct dirent 结构体， 包含一个名字和一个 i 节点编号。这个名字最多有 DIRSIZ (14)个字符；如果比较短，它将以 NUL 作为结尾字符。dirlookup 用于查找目录中指定名字的条目。函数 dirlink 会写入一个新的目录条目到目录 dp 下，用指定的名字和 i 节点编号。namei 解析 path 并返回对应的i 节点。函数nameiparent 是一个变种。它在最后一个元素之前停止，返回上级目录的 i 节点并且把最后一个元素拷贝到 name 中。这两个函数都使用 namex 来实现。namex 用于计算路径解析从什么地方开始；

1. 阅读文件file.h，file.c。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？
2. 阅读文件sysfile.c。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。

## 内容二：参考文献

1. 陈向群. 文件管理方案课件［Z］.
2. Bruce Molay. Unix/Linux编程［M］. 北京：清华大学出版社，2004．
3. Andrew S.Tanenbaum.现代操作系统［M］. 北京：机械工业出版社，2017．
4. Russ Cox, Frans Kasshoek, Robert Morris. XV6中文版［EB/OL］.