操作系统A 第四次阅读报告 xv6-文件系统

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。**

**超级块，** 文件系统中第一块被称为超级块。这个块存放文件系统本身的结构信息，比如每个区域的大小，未被使用的磁盘块的信息。

**i节点，** 超级块的下一个部分就是i节点表，文件系统中的每个文件在该表中都对应一个i节点。i节点是固定长度的记录项，它包含有关文件的大部分信息。Linux文件系统使用 索引节点i 来记录文件信息，索引节点是一个结构，用固定长度，它包含了一个文件的长度、创建及修改时间、权限、所属关系、磁盘中的位置等信息。

**数据块，** 文件的内容保存在这个区域上，磁盘上所有块的大小都一样，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，并把磁盘块的分配情况记录在文件的i节点中的磁盘序列表中。保存普通用户文件的磁盘块就是数据块。

**目录块，** 目录以文件的形式保存在一定的磁盘区域上，保存目录文件的磁盘块就是目录块。

**间接块，** 当存储大文件时就要用到间接块。UNIX文件系统的索引表有15个索引表项，后三个表项用来索引一级索引表，二级索引表和三级索引表，当文件大小超过12个块大小时，需要查找多级索引表，这些一、二、三级索引表所在的磁盘块称为间接块。

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**

**ide.c实现了一个基于程序控制I/O的硬盘驱动程序（PIO-based）。**

该驱动程序首先定义了几个磁盘状态的代号IDE\_BSY、IDE\_DRDY、IDE\_DF、IDE\_ERR、IDE\_CMD\_READ、IDE\_CMD\_WRITE。它维护了一个磁盘块缓冲区的队列ide\_queue，指向当前正在读写的一个磁盘块，ide\_queue->qnext指向下一个要访问的磁盘块，同时，为了实现磁盘访问的同步，该驱动还维护一个自旋锁ide\_lock，当访问磁盘时需要验证该锁。

**ide.c实现的主要函数及其功能：**

**int ide\_wait\_ready(int check\_error)：** 该函数用于等待磁盘准备完毕。该函数内部进行while循环，直到磁盘可用，随后根据是否发生错误返回0或-1。如果磁盘一直不可用，则该函数将阻塞直到磁盘准备完毕。

**void ide\_init(void)：** 初始化磁盘驱动程序，包括为锁ide\_lock赋初值，然后调用ide\_wait\_ready函数等待磁盘准备就绪，随后检查disk 1，并返回disk 0。

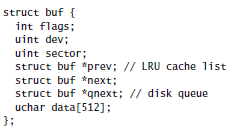
**void ide\_start\_request(struct buf \*b)：** 开始缓冲区b的磁盘访问请求，调用者为实现访问同步，必须事先获得锁ide\_lock。

**void ide\_intr(void)：** 磁盘驱动程序中的中断处理函数，用锁ide\_lock实现同步。该函数向内存缓冲区写入数据，并唤醒等待该数据的进程，然后根据qnext指针找到下一个要访问的硬盘快，调用ide\_start\_request函数访问下一个磁盘块。

**void ide\_rw(struct buf \*)：** 将修改的数据写回磁盘。检查B\_DIRTY位，如果该位为真，则将内容写回磁盘，并清除B\_DIRTY并将B\_VALID置1；如果B\_VALID未置1，则从磁盘中读取数据，写入缓冲区。

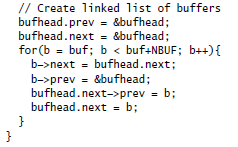
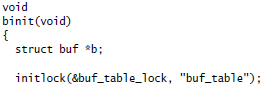
1. **阅读文件 buf.h，bio.c。了解 XV6 文件系统中 buffer cache 层的内容和实现。描述 buffer 双链表数据结构及其初始化过程。了解 buffer 的状态。了解对 buffer 的各种操作。**

xv6将硬盘中的每个分区编号为各种块，每块512Byte，磁盘读写总是以块为单位。

**xv6使用结构buf来代表磁盘块数据的缓冲区的表示**（位于buf.h文件中）：

**xv6的buffer以双链表的形式进行维护，该双链表的数据结构见上面的代码中的buf结构，并用LRU来实现缓存替换。**在bio.c中分配了一个双向链表bufhead维护着块的使用频率，按照最近使用的顺序来组织，表头是最近使用的块，表的尾部是最早访问的磁盘块。

**buf的初始化由binit函数实现，**binit函数实现如下：



**binit首先初始化了磁盘缓冲区的锁buf\_table\_lock，为了保证同步访问。然后初始化缓冲区的双向链表，因为初始化的时候尚未有元素，于是next和prev指针均指向表头元素bufhead，然后，依次将存在的磁盘缓冲区加入链表中，并修改他们的next和prev指针，形成双向链表结构。**

**一个buffer有三种状态**，这部分信息由buf.h定义：**B\_VALID** 意味着这个缓冲区拥有磁盘块的有效内容。**B\_DIRTY** 意味着缓冲区的内容已经被改变并且需要写回磁盘。**B\_BUSY** 意味着有某个内核线程持有这个缓冲区且尚未释放。

**bio.c实现了bget，bread，bwrite，brelse接口**，对buffer进行操作：

**bget：**bget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，则在缓冲区中找到一个无效的块缓冲区并返回，由bread调用iderw来将数据读入内核。如果缓冲区满，bget将panic“no buffer” 。

**bread：**bread从磁盘中取出一块放入缓冲区，并返回该缓冲区。

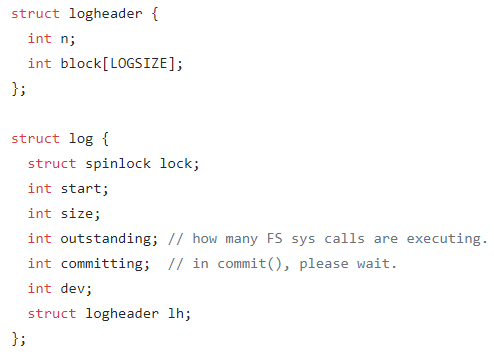
**bwrite：**bwrite把缓冲区中的一块写到磁盘上指定的地方。并修改该缓冲区的状态码。

**brelse：**brelse释放一个缓冲块，将这个缓冲块从双向链表中删去，然后修改他的状态码。

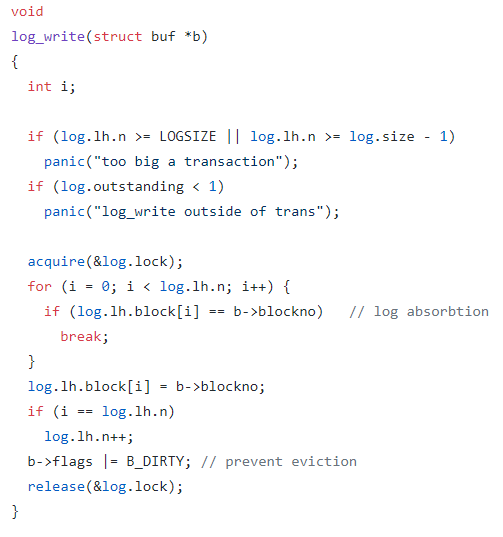
1. **阅读文件 log.c。了解 XV6 文件系统中的 logging 和 transaction 机制。**

xv6使用了日志式文件系统来确保写操作不会导致文件系统的破坏，进程的写操作像一种“原子”操作。 xv6使用了非常严格的日志读写来使读写操作要么完全完成，要么完全未完成。所有的读写操作首先都会写入磁盘中存放日志的区域，只有当真正的读写操作完成后才会使日志失效，这样，就算任何过程中断电或者其他原因导致系统崩溃，文件系统的组织结构都不会损坏，结果是要么操作完全完成，要么都未完成。

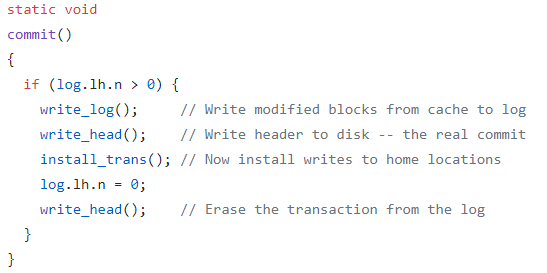
xv6在硬盘中的日志有一个初始快和数据块，初始快包括一个数组，数组的值为对应数据块的内容应该写入文件系统中的哪一块，初始快还有当前有效数据块的计数。在内存中同样要一样的结构来存储数据。



通过这种方式，bwrite可以使用log\_write替代，当修改了内存中的块缓冲区后，log\_wirte同时在block数组中记录这个块需要写到磁盘中的哪一块，但是没有立即写入，当调用commit的时候，调用log\_write写入日志区域中，并调用write\_head更新初始快，然后调用install\_trans真正地更新文件系统，此时，发生崩溃都会导致日志有非零的计数，以便重启后再次进行写操作，最后将计数变量置零使日志失效并更新日志初始快。



通过log\_write写入磁盘时，数据并不会立即写入磁盘，只有当调用commit来提交日志时，磁盘操作才会正式开始磁盘操作。xv6日志读写支持并发操作，当要写操作时，调用begin\_op，结束时调用end\_op，begin\_op检查日志是否正在提交，如果正在提交则睡眠当前进程，如果不在提交则增加操作次数，end\_op减少操作次数，当没有任何进程正在操作log时，调用commit提交日志。

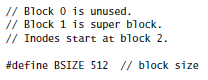


1. **阅读文件 fs.h，fs.c。了解 XV6 文件系统的硬盘布局。**

**硬盘布局如下：**

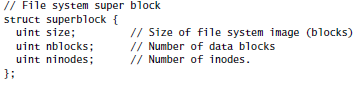
**[boot block || super block || log || inode blocks || free bit map || data blocks ]**

**在fs.h头文件中可以看到，xv6以块为单位组织磁盘，每个块的大小为512字节。块号为0的块是不被使用的，块号为1的块是超级块，从2号块开始是I节点的区域。之后是数据区，包括目录文件和用户文件。**



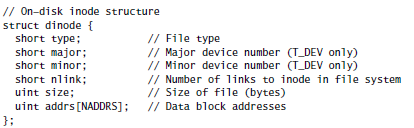
（fs.h中对硬盘布局的描述）

**超级块**组成如下：



超级快中包括文件系统的大小，数据块的数量和I节点的数量的信息。

**I节点（Inode）**组成如下（这里特指硬盘中的i节点结构）：



其中包括文件类型，设备号，与文件系统i节点的连接数，文件大小和数据块地址序列的信息

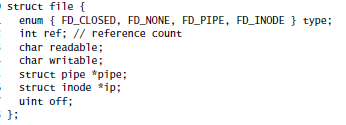
用户文件和目录文件的结构在问题6.中有详细描述，故不在此处展开。

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？**

在file.h的file结构中可以看到**文件类型由变量type指定**，共有三类：

**FD\_NONE、FD\_PIPE、FD\_INODE。**

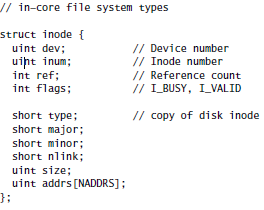
**文件相关的数据结构为struct file**，在file.h中定义



结构中包含文件类型，文件引用数，可读标记，可写标记，文件I节点的指针，pipe的指针等信息。

**i节点相关的数据结构为 struct inode和struct dinode**

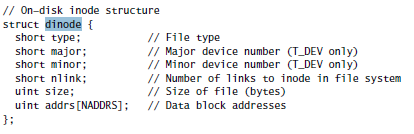
**其中 inode在fsvar.h中定义**：



包含设备号，i节点号，引用数，状态码，磁盘i节点的信息，大小，地址序列等信息。

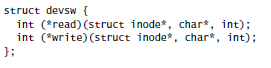
**另外，dinode**在fs.h中定义，**这是磁盘上的i节点的数据结构**：

其中包括文件类型，设备号，与文件系统i节点的连接数，文件大小和数据块地址序列的信息，其实现见下图：



**设备相关的数据结构在dev.h中定义了devsw结构：**

**结构中包含两个函数指针，分别对应对设备的读和写**



**xv6的文件操作：**

主要由file.c中的若干函数实现，下面分别简介这些文件操作的内容和函数：

**struct file\* filealloc(void)：**为文件分配一个文件结构（file结构），并更新其中的内容。

**void fileclose(struct file\*)：**关闭文件。

**struct file\* filedup(struct file\*)：**增加文件的引用数。

**void fileinit(void)：**初始化文件，初始化一个访问文件的锁。

**int fileread(struct file\*, char\*, int n)：**从文件中读信息，地址采用内核地址。

**int** **filestat(struct file\*, struct stat\*)：**获取文件的元数据。

**int filewrite(struct file\*, char\*, int n)：**向文件中写信息，地址是内核地址。

**XV6最多支持系统打开100个文件，其中每个进程可打开16个文件，**活跃的i节点数量最多为50个，参见**param.h**中的定义：



1. **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。**

在sysfile.c中实现了以下系统调用，这些系统调用与文件系统相关，因此与其他系统调用分开，单独成立一个文件：

**sys\_read：**读文件

**sys\_write**：向文件中写入

**sys\_dup：**增加文件的引用数

**sys\_close**：关闭文件

**sys\_fstat：**获取文件元数据并返回

**sys\_link：**创建一条新的指向已存在的的i节点的链接路径