XV6 文件系统

by 1801210840 姜慧强

XV6 文件系统结构

XV6 是基于 Unix 的，所以其文件系统与 Linux 较为类似，也是 Block 0 不使用，Block 1 是超级块，Block 是 I 节点区，之后才是根目录区，普通文件区。该部分内容定义在 fs.h 中。

XV6 中一个 Block 大小是 512Bit。

超级块定义了文件系统拥有多少数据块，拥有多少数据块，拥有多少节点，拥有多少日志块。

```
// File system super block
struct superblock {
  uint size;         // Size of file system image (blocks)
  uint nblocks;      // Number of data blocks
  uint ninodes;      // Number of inodes.
  uint nlog;         // Number of log blocks
};
```

还定义了磁盘 i 节点结构，其中记录了文件类型，主要设备数量，次要设备数量，文件的硬链接数量，文件大小，块数据地址。

```
// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+1];   // Data block addresses
};
```

XV6 文件系统实现接口

相对应的在 fs.c 文件中定义了一系列有关上述 struct 的操作函数。

通过 readsb 来读取超级块中数据内容，具体是通过 buffer cache 来读取第 1 块 block 中内容。

通过 bzero 来对数据块做清零作业，同样通过 buffer cache 来获取相应的第 n 块 block，然后通过 memset 设置为 0。

同时写入 log 日志。

通过 balloc 来分配一个 0 块。遍历超级块，如果查询到有空闲位置，则调用 bzero 分配相应的 zero block。

通过 bfree 释放一个块数据，同样读取超级块信息，然后设置 buffer cache 内容。

其次，通过 spinLock + inode 来实现一个同步 i 节点 icache。

和之前相类似，有对应的分配函数，初始化函数，get 函数，更新状态函数，对给定的 i 节点上锁，对指定的 i 节点释放锁，释放所分配的 i 节点。

Data 之间的联系通过 Block 存储，如果没找到相应的关系，则通过 bmap 来分配。

另外还提供阶段 i 节点的函数 itrunc。

拷贝函数，将 i 节点信息拷贝给 stat.

QA

UNIX 文件系统中主要组成部分

Unix 文件系统 由超级数据块，i 节点，数据块，目录块，间接块等部分组成。

超级数据块，保存的是一些文件系统的全局变量，文件系统的元信息，包括文件系统总块数，数据块块数，i 节点数，日志块数。超级数据块占用第 1 号 Block。

I 节点区存放文件具体分配的 Block 信息。

紧接着 I 节点区的是空闲块位图，用来维护空闲区域内容。

在 i 节点之后的才是最多的数据块，在这之中保存了文件和目录内容。在磁盘的最后是日志块，用来记录日志信息。

IDE

XV6 中数据存储在一块 IDE 磁盘中，XV6 通过块缓冲读写 IDE 硬盘,这是一个同步磁盘，同样利用自旋锁 spinLock 来完成.

保证只有一个内核进程可以修改磁盘块。

其实现了 idewait，ideinit，idestart，ideintr，iderw 五个函数，分别用作等待 IDE 磁盘出于就绪状态，初始化 IDE 磁盘，开始请求，中断处理，同步块缓存区和 IDE 磁盘。

Buffer cache

块缓存区在 XV6 中实现了两个任务

1. 同步对磁盘的访问，使得对于每一块，在同一时刻中只有一份 buffer 在内存中，并且只有一个内核进程能使用这份 buffer
2. 缓存常用的 Block，提升文件系统性能

具体而言，通过 bread 和 bwrite 两个函数来完成 IDE 和 Buffer cache 的信息同步。

bread 从 IDE 磁盘中取出一块放入缓冲区，bwrite 把缓冲区的一块写入磁盘中争取的地方。当内核程序处理完一个缓冲块之后，需要调用 brelse 释放它。

Buffer Cache 也是利用 SpinLock 来实现互斥过程，实现对块缓冲块的互斥访问。但一个内核进程以及使用了缓冲块，则其他使用缓冲块的进程则会被堵塞。

Buffer Cache 通过 LRU 来完成调度管理。

块缓冲区通过双向链表来管理缓冲区，binit 从一个静态数组 buf 中构建一个有 NBUF 元素的双向链表，而所有对块缓冲区访问都是通过链表而并非静态数组。

块缓冲区有三种状态

- B_VALID 缓冲区拥有磁盘块的有效内容
- B_DIRTY 缓冲区内容被修改，需要写回磁盘
- B_BUST 缓冲区被某个进程占用且未被释放

bread 函数调用 bget 获得指定扇区的缓冲区，如果缓冲区 Miss 则在从 Disk load 之后还会写入 Buffer Cache，这步通过调用 iderw 来实现。

而 bget 函数扫描缓冲区链表，通过给定的扇区好，derive id 找到对应的缓冲区。如果找到一个不处于 B_BUSY 状态，bget 就会设置它的 B_BUSY 位并且返回。

如果满足条件的缓冲区被使用了，bget 就会睡眠等待其被释放。

日志块

参考 DBMS 等系统，为保证一致性和可恢复性，XV6 的文件系统增加了日志层。

当文件系统发生写操作，因为写磁盘并非原子操作，如若发生崩溃，则会出现不一致现象。

在 XV6 中一个系统调用并不直接对磁盘上的文件系统进行写操作。而是通过一个对把写磁盘的操作包装成一个例子写在磁盘中。 当系统调用吧所有写操作都写入了日志，则会把所有日志提交到磁盘上，代表完成一个完整的操作。与此同时，系统调用则会解析日志内容，并执行相应的磁盘操作。

当写操作完成之后，则会删除相应的日志文件。

这种做法常见于 DBMS、分布式文件系统中，通过 log 来达成一致性。

在 XV6 中，日志块处于磁盘末端，包含一个起始块，紧接着一连串数据块，起始块包含一个扇区号数组，每一个对应日志中的数据块，类似于 I 节点区，除此之外还包含日志数据块的计数。

XV6 对于每个系统调用中一系列写操作称之为 Session。每一个 Session 是原子性的，但 XV6 不支持并发 Session。并发 Session 容易产生不一致现象。

在 DBMS 中类似的操作，需要判断可串行化，具体而言判断视图可串行化，冲突可串行化等。XV6 中对多阶段原子性写操作的时间不能支持并行 Session 的要求。

而对于只读的 Session 是可以并发操作的，通过 i 节点锁来实现原子性。

xv6 允许只读的系统调用在一次会话中并发执行。i 节点锁会使得会话对只读系统调用看上去是原子性的。

xv6 使用固定量的磁盘空间来保存日志。系统调用写入日志的块的总大小不能大于日志的总大小。对于大多数系统调用来说这都不是个问题，但是其中两个可能会写大量的块：write 和 unlink。写一个大文件可能会写很多的数据块、位图块，以及 i 节点块。移除对一个大文件的链接可能会写很多的位图块以及一个 i 节点块。xv6 的写系统调用将大的写操作拆分成几个小的写操作，使得被修改的块能放入日志中。unlink 不会导致问题因为实际上 xv6 只使用一个位图块。

文件

文件 struct 定义在 file.h 文件中。其中有 FD_NONE, FD_PIPE, FD_INODE 三种状态。

在 file 结构体重还定义了被参考计数，是否可读，是否可写，i 节点，pipe 管道。

同时 inode 也是在 file.h 中定义的,在这之中定义了设备 ID，参考计数，flags,节点 Link 数，类型等等。

```
// in-memory copy of an inode
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  int flags;          // I_BUSY, I_VALID

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+1];
};
```

同样 XV6 也是定义一个 同步版本的 ftable 来控制互斥的访问。

XV6 最多支持打开 100 个文件，10 个设备

systcall

XV6 定义了 21 种系统调用。

除了常规的系统调用之外，因为这是 Unix 的文件系统，会有 i 节点和软链接，而建立连接，和释放连接都是通过系统调用来实现的。

sys_link 增加 ip->nlink 计数。然后调用 nameiparent(new)来寻找父目录，并创建一个目录项指向旧的 I 节点。

相对而言，create 则是从一个新的节点中进行创建。首先用 O_CREATE open 一个文件创建一个新的普通文件。然后调用 nameiparent(new)寻找上级目录的 i 节点，并调用 dirlookup 来检查同名文件是否存在。如果文件名不重复则调用 ialloc 来分配一个新的 i 节点。如果 create 一个目录文件，则还会初始化., ..两个目录项。

Mkdir 则创建一个新的目录文件，mkdev 创建一个新的设备文件。