文件系统初始化

Unix V6++ 文件系统设计分析

关于磁盘、扇区和盘块

对于机械硬盘来说，我们一般可以通过磁头、柱面、物理扇区三个值来描述磁盘，但这种方式较为复杂，其实也可以直接规定出扇区的线性地址，也就是说我们将所有的扇区按磁头、柱面、扇区号等进行排序，给每个扇区一个唯一确定的标号，用这个标号来描述扇区显得容易许多。

在本实验中，我们需要通过一个img文件，来模拟一个物理磁盘，在该磁盘文件内部，扇区的排列方式就是线性的。我们在该文件内模拟文件系统的构成，供操作系统使用。

每个扇区的固定大小是512字节，在本实验中，我将img文件的大小设置为10800KB，即21600个扇区，可用的扇区标号为[0, 21599]。

扇区是操作系统一次性读取操作的最小单位，然而对于某些操作系统，512字节还是太小了，所以也就有了盘块的概念。盘块由多个扇区组成，可由操作系统一次性读取。例如Linux盘块大小为4KB，是8个扇区。

Unix V6++ 文件系统的扇区分布

信息	bootloader	kernel	superblock	inode	data	swap
位置（扇区号）	[0, 0]	[1, 199]	[200, 201]	[202, 1023]	[1024, 17999]	[18000, 20159]

bootloader

bootloader是在操作系统内核运行之前运行。可以初始化硬件设备、建立内存空间映射图，从而将系统的软硬件环境带到一个合适状态，以便为最终调用操作系统准备好正确的环境。通常CPU加电后，首先进入BIOS程序。BIOS程序完成准备工作后，将磁盘中的第一个扇区加载到内存中，执行其中代码，该部分代码被称为boot程序。boot程序会加载loader程序，后者将完成其余准备工作，加载内核，跳转进入内核代码。这二者被合成为bootloader。Unix V6++中并无loader程序，仅通过一个boot完成基本设置。

kernel

内核二进制文件。该文件需要用特殊方式编译，并通过对链接命令或链接脚本的设置，将进入点放置在指定偏移位置。

superblock

描述文件系统结构的重要结构，记录有硬盘大小、交换区位置等信息，完成空盘块管理、空inode管理等任务。

在硬盘中，前200个盘块和swap区都不由文件系统直接管理。文件系统管理的是superblock, inode区和data区。

```
public:
    int s_isize; /* 外存Inode区占用的盘块数 */
    int s_fsize; /* 盘块总数 */

    int s_nfree;     /* 直接管理的空闲盘块数量 */
    int s_free[100]; /* 直接管理的空闲盘块索引表 */

    int s_ninode;     /* 直接管理的空闲外存Inode数量 */
    int s_inode[100]; /* 直接管理的空闲外存Inode索引表 */

    int s_flock; /* 封锁空闲盘块索引表标志 */
    int s_ilock; /* 封锁空闲Inode表标志 */

    int s_fmod;      /* 内存中super block副本被修改标志，意味着需要更新外存对应的Super Block */
    int s_ronly;     /* 本文件系统只能读出 */
    int s_time;      /* 最近一次更新时间 */
    int padding[47]; /* 填充使SuperBlock块大小等于1024字节，占据2个扇区 */
};
```

superblock的结构如上图所示。

- s_isize: 记录inode区占用的空间，单位是盘块，这里应当是822
- s_fsize: 记录整个盘块的盘块总数，应该是21600
- s_nfree: 记录直接管理的空闲盘块数量
- s_free: 记录直接管理的空闲盘块索引表
- s_ninode: 记录管理的空闲inode数量
- s_inode: 记录管理的空闲inode索引表
    - s_inode和s_ninode形成的栈结构总共可以管理100个空闲inode，当我们申请一个新的inode时，若s_ninode大于0，说明此时还有空余inode，直接将s_ninode - 1编号的inode分配，s_ninode -= 1; 若s_ninode小于等于0， 将前往inode区域重新寻找空闲inode来填充整个栈，直到对inode区搜索完毕，或直接管理的空闲inode重新达到100个
- s_flock: 空闲盘块锁
- s_ilock: 空闲inode锁
    - 这两个锁是为了防止在并发情况下出现冲突
- s_fmod: 代表内存中的superblock是否被修改，如果被修改需要更新外存中的superblock
- s_ronly: 代表该文件系统是否是只读的
- s_time: 代表文件系统最后一次更新时间。记录unix时间戳，以秒为单位

inode

每个inode占用64个字节，一个盘块(512字节)可以存储8个inode结构。inode是用来表示文件的重要结构。这个文件可以是指普通文件，也可以指文件夹、块设备和字符设备。其内记录了文件大小、文件类型等基本信息。并且其内部的二级索引机制可以链接6 + 128 * 2 + 128 * 128 * 2 = 33030个盘块，令操作系统可以存储约16MB的文件。

```
public:
    unsigned int d_mode; /* 状态的标志位，定义见enum INodeFlag */
    int d_nlink;         /* 文件联结计数，即该文件在目录树中不同路径名的数量 */

    short d_uid; /* 文件所有者的用户标识数 */
    short d_gid; /* 文件所有者的组标识数 */

    int d_size;     /* 文件大小，字节为单位 */
    int d_addr[10]; /* 用于文件逻辑块好和物理块好转换的基本索引表 */

    int d_atime; /* 最后访问时间 */
    int d_mtime; /* 最后修改时间 */
```

每个inode结点可以存储10个盘块信息，即位于d_addr结构中。

其中前6个结点直接指向数据盘块，每个盘块大小为512KB，6个结点一共可以存储3KB大小的数据文件。如果后面的4个结点也采取这种方式，那么每个inode只能指向5KB大小的数据空间，这显然是不可接受的。于是在后面分别采取了一级索引和二级索引的结构。

第7-8结点采用了一级索引结构，他们也分别指向一个盘块，不过这个盘块内部存储的并不是数据，而是像前6个结点一样的索引，也就是说一个盘块大小为512字节，里面存储了128个4字节的小索引，这样以来这两个结点可以映射到的数据空间大小为2 * 128 *512 = 128KB。

不过有了前两个索引结点还是不能满足我们的要求，于是最后两个结点，9-10结点采用了二级索引结构，他们指向的盘块中存储的是一级索引结构，也就是说在大小为512字节的盘块中存放了128个一级索引，每个一级索引可以管理的数据空间大小为128 * 512 = 64KB， 那么每个二级索引结点可以管理的数据空间大小为 128 * 64KB = 8MB，两个二级索引结点一共可以管理16MB的空间。

如此以来，一个文件的最大大小约为16.1MB。

由于一级索引块和二级索引块会占用数据块大小，所以一个文件真正的存储空间是可能大于其数据内容大小的。

data

存放文件数据的盘块，也可能存储的有二级索引结构中的索引表。

swap

该部分为Unix V6++操作系统的盘交换区。

目录文件结构

树形文件结构是依靠文件夹实现的。文件夹自身也是一个文件。文件夹内部的每32个字节表示文件夹里的一个文件。其中，前4字节是inode编号，指向该文件的inode。后28字节是文件名。文件名结束之后的部分用0填充。

这样以来，每个文件夹可以存储16MB/32B 约 50 万个文件

superblock内的空闲盘块管理模式

在磁盘中，从1024号盘块到17999号盘块都被作为数据盘块使用。如果采用类似管理空闲inode的方式管理空闲盘块，搜索速度将会变得很慢。另外盘块上没有该盘块是否被使用的标记，使得类似管理inode的方式根本无法使用。为此，文件系统内采用了一种“栈中栈”的结构来管理空闲盘块。

superblock内的s_nfree表示superblock内直接管理多少个空闲盘块，s_free记录空闲盘块号。

假设当前s_nfree = 99, superblock直接管理99个空闲盘块，这些盘块的编号依次存储在s_free[0]到s_nfree[98]中。此时，如果我们释放一个盘块，比如释放的是3000号盘块，那么将3000记录到s_free[99]，使得s_nfree=100。

之后，如果继续释放盘块，比如3100号盘块，此时superblock内已经没有登记它的地方。于是，我们将s_nfree和整个s_free数组的值拷贝到第3100号盘块内，共404字节，令superblock的s_nfree = 1，s_free[0] = 3100。

当我们希望获取一个空闲盘块时，发现superblock直接管理的盘块只有一个，我们先记录下要分配的盘块号(s_free[0]),然后从该盘块内读取404字节信息，拷贝到superblock内，从s_nfree开始连续更新404字节的内容。

当然为了标记结尾，会使得最后一组盘块的s_free[0]为0，该组仅提供99个真实可用的数据盘块。