第 5 次操作系统实验报告

161810129 董世晨

一、 回答问题

1. 为何OS尚未切换到scheduler时，读写磁盘不能用中断的方式进行？

原因有很多。首先，在检查磁盘时还没有开启中断；其次，fs.c中使用中断来读写磁盘是使用sleep/wakup原语实现的，而这两个原语都依赖于scheduler调度，然而检查磁盘时还没有开始执行scheduler，故无法使用中断；最后最关键的是，检查磁盘时只有系统进程正在运行，没有其他进程，启动磁盘读写程序后，CPU没有其他进程可以调度，只能轮询，或者说，使用中断不会有任何性能提升。

这里我想记录一下我实现轮询磁盘读写的曲折道路。我的第一个思路是，使用一个全局变量指示是否正在检验文件系统，如果是，将fs.c中的所有基于acquiresleep和releasesleep修改为轮询的形式。经过很久的调试，发现ide.c中使用了sleep，而我认为很难将这段代码改成轮询的形式。最终借鉴了bootmain.c中的readsect函数和ide.c中的idestart函数实现了轮询读写磁盘。

2. 写出你遍历目录的伪代码

```
定义函数：fsckrec
参数：idx，表示检查第idx个dinode
伪代码：
    记录下已访问过第idx个dinode
    获取第idx个dinode，存到ip中
    若ip指向的不是一个文件夹，则退出
    获取ip指向的文件夹目录，存到dp中
    循环遍历dp中的所有非“.”或“..”的子项，记为item
        若item.inum不为0
            递归调用fsckrec(item.inum)
```

3. 你是使用全局变量存储inode，还是使用局部变量或者是用kalloc和kfree？

我使用kalloc实现了**带有延迟写入的LRU缓存**。

虽然实验要求的文件系统只有200个inode，实际上，我的笔记本的WSL子系统上有1600多万个inode资源，按照xv6的dinode大小需要近1GB的内存。因此，一次性读入所有dinode是不现实的。然而，如果每次只读入一部分的dinode，可能要重复读取很多次磁盘。综上，我为block实现了一个LRU缓存，使用较少的内存和较少的磁盘读写次数实现文件系统校验。

我选择一种比较简单的LRU缓存实现方式：使用**双向链表**存储节点，每次访问需要遍历搜索该链表，如果找到了，将该节点移动到链表头，如果没找到，则重用链表尾，进行磁盘读写，插入到链表头。详细代码见fsck.c中的lruget函数。

虽然说着简单，实际上使用纯C语言实现一个LRU需要考虑很多边界条件，如移动节点时需要判断是否是链表头或链表尾、未找到时重用尾节点还是使用新节点等。考虑所有的情况需要极大的细心和耐心。

由于需要修复inode需要对inode进行修改，我采取了**延迟写入**的技术。每当我修改LRU缓存中的内容之后，会将该block标记为dirty，而不是writethrough。每当重用尾节点或者释放LRU缓存时，判断该block是否为dirty，若是则进行写磁盘操作。

在LRU缓存的基础上，我对一些常用的函数进行了封装，以方便后续代码的编写。获取文件夹内容、间接指向的block、修改free bit map等都需要进一步读盘操作，这些读盘操作也是使用了LRU缓存的。

```
// 获取第 inode 个 inode
struct dinode *getinode(struct lrucache *cache, uint inode);
// 若 inode 指向文件夹，获取 inode 的所有 dirent
struct dirent *getdirents(struct lrucache *cache, struct dinode *inode);
// 获取 inode 间接指向的 block
static uint *getindirects(struct lrucache *cache, struct dinode *inode);
```

对于实验要求的只有200个inode的文件系统，我只使用了一个页框，也就是8个LRU节点。

4. 实验结果

proj5：修复了3、5、7、13这4个inode，释放了122个block。

proj5i：修复了20~26这7个inode，释放了7个block。

第一次运行make proj5：

```
xv6...
cpu0: starting 0
Running fsck...
fsck: inode 3 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 5 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 7 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 13 is allocated but is not referenced by any dir! Fixing... Done!
fsck completed: Fixed 4 inodes and freed 122 disk blocks.
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

第二次运行make proj5：

```
xv6...
cpu0: starting 0
Running fsck...
fsck completed: no problem found
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

第一次运行make proj5i：

```
xv6...
cpu0: starting 0
Running fsck...
fsck: inode 20 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 21 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 22 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 23 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 24 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 25 is allocated but is not referenced by any dir! Fixing... Done!
fsck: inode 26 is allocated but is not referenced by any dir! Fixing... Done!
fsck completed: Fixed 7 inodes and freed 7 disk blocks.
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

第二次运行make proj5i：

```
xv6...
cpu0: starting 0
Running fsck...
fsck completed: no problem found
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

（后面还有）

二、 补充细节

1. 如何修复错误的inode

释放block时，还需要将free bit map中对应的标记位修改为0。以下给出修复inode的伪代码。

```
定义函数：freeblock
参数：block，表示要修改free bit map的block编号
返回值：若修改成功，返回1；若无需修改，返回0
伪代码：
    读取该block的free bit map对应标记位所在的block，存到data中
    若该标记位为0，返回0
    修改该标记位为0
    返回1
```

```
定义函数：fixblock
参数：inode，表示要释放其指向的block的inode指针
伪代码：
    循环遍历该inode下的所有direct块，记为addr
        如果addr不为0，则调用freeblock(addr)
    如果该inode下的indirect块被使用（不为0），记为addr
        调用freeblock(addr)
        循环遍历indirect块中所有块，记为addr
            如果addr不为0，则调用freeblock(addr)
```

```
定义函数：fixinode
说明：释放所有有问题的inode
伪代码：
    循环遍历所有inode，记为ip
        若ip没有被访问过，且ip.type不为0
            将ip.type改为0
            调用fixblock(ip)
```

2. 除了使用LRU缓存，还能做什么来节省内存

1) 我使用bitmap来记录200个inode是否已访问，只需要占用25字节的栈空间
2) 我的代码中**没有使用任何全局变量**，也就是所有内存使用完之后都会释放。即使是LRU缓存的控制变量也使用了栈空间。

（后面还有）

三、 后记

让我以 git 提交记录作为这 6 次实验的后记：