Lab6

姓名: 陈锐林, 学号:21307130148

2023 年 12 月 5 日

实验 1

一、实验思路

1. 这题思路比较简单，首先要完善对 NDIRECT、NINDIRECT 的定义和进行 inode,dinode 的修改；主要是为了和目标中扩展一个块作对应。其中 NINDIRECT 应该是 (BSIZE/sizeof(uint))，也就是要求的 1024/4=256；还要记得修改 MAXFILE 的定义，改为 NDIRECT + NINDIRECT + NINDIRECT * NINDIRECT。

2. 接着主要是完善 bmap 函数，参考已有的 bmap，能看出要用以下函数：balloc 用于分配，bread 进入目录，log_write 进行写 log，以及最后 brelse 先前 bread 的目录。接着参照对于但间接数据块的流程，完善两级的。首先加载 NDIRECT+1 的块 (如果未分配要补上)，接着记载目录进入下一级；未分配还要补上；最后才进入二级，并重复操作。其中三次取的索引就应该是：NDIRECT+1，bn / NINDIRECT，bn% NINDIRECT。

3. 最后是在 itrunc 函数中完善释放块的任务。可以看到很类似的，对于 1 级的，就是进入目录一一释放；那么 2 级的就是在 1 级的基础上，先不释放 1 级，再依次遍历 2 级数据并释放。

二、具体实现

1. 完成预定义：NDIRECT，NINDIRECT，MAXFILE 的定义如下，为了对应缩小了 1 的 NDIRECT，将 inode,dinode 中的 addrs 改为”addrs[NDIRECT + 2]”（未展示）.

```
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT*NINDIRECT)
```

2.bmap 函数：仿照前面的 1 级处理，完善” 实验思路 2.” 的 bmap 函数。根据文件系统的索引规则，确定两次读取的内容 bp1 和 bp2；未分配时及时分配即可。

```
bn -= NINDIRECT;
if(bn < NINDIRECT * NINDIRECT){
  if((addr = ip->addrs[NDIRECT+1]) ==0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
  bp1 = bread(ip->dev,addr);
  a = (uint*)bp1->data;
  if((addr = a[bn / NINDIRECT]) == 0){
    a[bn / NINDIRECT] = addr = balloc(ip->dev);
    log_write(bp1);
  }
  bp2 = bread(ip->dev,addr);
  a = (uint*)bp2->data;
  if((addr = a[bn % NINDIRECT]) == 0){
    a[bn % NINDIRECT] = addr = balloc(ip->dev);
    log_write(bp2);
  }
  brelse(bp2);
  brelse(bp1);
  return addr;
}
```

3.itrunc 函数：完善释放的函数，当 ip-addr 指向 2 级的块是进入处理。内部使用两个 for 循环进行递归，凡是分配了的都要及时 bfree；最后每次退出要及时 brelse；最外部完成对直接数据块的释放。

```
if(ip->addrs[NDIRECT + 1]){
  bp = bread(ip->dev,ip->addrs[NDIRECT + 1]);
  a = (uint*)bp->data;
  for(int i = 0;i < NINDIRECT;i++){
    if(a[i]){
    bp2 = bread(ip->dev,a[i]);
    uint* a2 = (uint*)bp2->data;
    for(j = 0;j < NINDIRECT;j++){
      if(a2[j])
        bfree(ip->dev,a2[j]);
    }
    brelse(bp2);
    bfree(ip->dev,a[i]);
    }
  }
  brelse(bp);
  bfree(ip->dev,ip->addrs[NDIRECT+1]);
}
```

实验 2

一、实验思路：

1. 准备工作，包括:(1) 系统调用老生常谈的 user/usys.pl、user/user.h、kernel/syscal.h、kernel/syscal.c，不再赘述；(2)kernel/stat.h 中添加新的文件类型 T_SYMLINK，为 4(顺延)；(3)kernel/fcntl.h 中添加新标志 O_NOFOLLOW，考虑到传递给 open 时使用 or 操作，所以必须是独立的；顺延下取 0x800。

2. 这里打算直接在 sysfile.c 里实现 symlink，就不设空了。参考 sysfile 其他函数和 xv6 的资料，能看到，开始/结束文件系统操作需要调用函数 nameiparent() 获取父目录信息，begin_op() 和 end_op()，ilock() 用于锁定 inode，ialloc() 用于分配 inode，writei() 用于向 inode 写入链接信息，dirlink() 用于在父目录建立链接，iupdate() 用于更新 inode，以及完成操作后应该要用 iunlockput() 释放锁并增加计数，用 iput() 增加计数。所以 symlink 的过程应该是这样的：通过 argstr 获得目标路径和链接路径；nameiparent 获取目标路径的父目录和链接名称：ialloc 分配一个 inode 并 writei 写入目标路径；dirlink 链接到父目录；最后更新连接数。

3. 更改 open 函数。根据已有的模板，主要是多添一条，在 ip->type == T_SYMLINK && omode != O_NOFOLLOW) 时进入该处理。之后我们应该打开链接的地址；考虑到多次链接的情况，需要包装一个循环解决这个问题，循环终止条件就是 (1) 有循环链接 (简单标识为寻找次数大于 12) 或者 (2) 不再是 T_SYMLINK 类型了。具体的实现利用 readi 函数读取当前 inode 对应目标路径和 namei 获取新的 inode。while(...)readi(...),namei(...)。

二、具体实现

1. 此处省略。

2.(1) 读取参数；(2) 获得父目录信息；(3) 分配一个 inode 并写入目标路径:

```
if(argstr(0,target,MAXPATH) < 0 ||
 argstr(1,path,MAXPATH) < 0)
  return -1;
begin_op();
if((dp = nameiparent(path,name)) == 0){
  end_op();
  return -1;
}
```

(a) 读参数

```
if(writei(sp,0,(uint64)target,0,sizeof(target))
 != sizeof(target)){
  iunlockput(sp);
  iput(dp);
  end_op();
  return -1;
}
```

(b) 得父目录信息

(3) 链接到父目录；(4) 完成计数更新。注：begin_op() 和 ilock() 等因为篇幅原因没有放入。

```
if(dirlink(dp,name,sp->inum) < 0){
  iunlockput(dp);
  iunlockput(sp);
  end_op();
  return -1;
}
```

(a) 读参数

```
iunlockput(dp);
sp->nlink++;
iupdate(sp);
iunlockput(sp);
end_op();
return 0;
```

(b) 得父目录信息

3. 修改 open 函数，(1)open 函数中特判如下，symlink_ip 函数是包装出去实现的：

```
if(ip->type == T_SYMLINK && omode != O_NOFOLLOW){
  struct inode *newip = ip;
  if(symlink_ip(newip,&ip)!= 0){
    end_op();
  return -1;
  }
}
```

(2)symlink_ip 函数中，主体循环如下。主要是如上利用了 readi 读取当前 inode 的目标路径和 namei 获取新的 inode。

```
do{
  char target[MAXPATH];
  readi(p,0,(uint64)target,0,MAXPATH);
  iunlockput(p);
  if((p=namei(target))==0){
    return -1;
  }
  count++;
  ilock(p);
}while(count < 12 && p->type == T_SYMLINK);
```

★ 测试结果 (make grade 后的结果，两个实验一起的)

```
== Test running bigfile ==
$ make qemu-gdb
running bigfile: OK (153.4s)
== Test running symlinktest ==
$ make qemu-gdb
(0.9s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (212.2s)
== Test time ==
time: OK
Score: 100/100
```