OS Lab 6

file system

王世炟

PB20151796

December 11, 2022

Part 1: 实验要求

在本实验中，我们要修改 xv6 的文件系统，以实现某些功能。

Part 2: Large files

2.1 问题简述

第一个任务是通过实现二级索引增大 xv6 所支持的文件大小。

2.2 知识基础

xv6 中的 inode 有 12 个直接索引（直接对应了 data 区域的磁盘块），1 个一级索引（存放另一个指向 data 区域的索引）。因此，最多支持 12 + 256 = 268 个数据块。如下图所示：

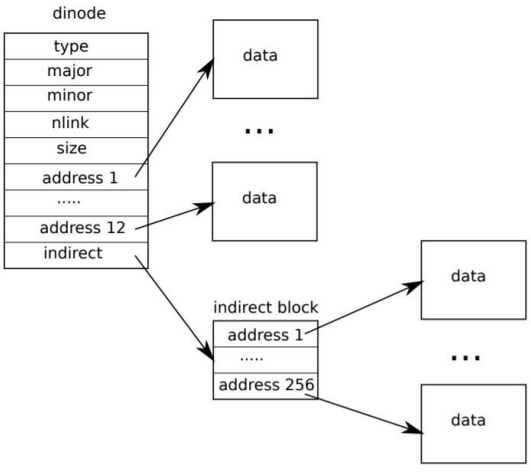

Figure 8.3: The representation of a file on disk.

图 1

2.3 实验步骤

(1) 更改内存块结构

首先，kernel/fs.h 文件中减小 NDIRECT 的值，为二级索引留一个位置:

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

// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEVICE only)
  short minor;          // Minor device number (T_DEVICE only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+2];   // Data block addresses
};
```

上面的是磁盘中的 inode 结构，还需要在 kernel/file.h 中更改内存中的 inode 结构:

```
// in-memory copy of an inode
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock; // protects everything below here
  int valid;          // inode has been read from disk?

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+2];
};
```

(2) 扩充 bmap() 函数

仿照一级索引，写出二级索引，在 kernel/fs.c 中添加代码::

```
static uint
bmap(struct inode *ip, uint bn)
```

```
3  {
4    uint addr, *a;
5    struct buf *bp;
6
7    if(bn < NDIRECT){
8      if((addr = ip->addrs[bn]) == 0)
9        ip->addrs[bn] = addr = balloc(ip->dev);
10     return addr;
11   }
12   bn -= NDIRECT;
13
14   if(bn < NINDIRECT){
15     // Load indirect block, allocating if necessary.
16     if((addr = ip->addrs[NDIRECT]) == 0)
17       ip->addrs[NDIRECT] = addr = balloc(ip->dev);
18     bp = bread(ip->dev, addr);
19     a = (uint*)bp->data;
20     if((addr = a[bn]) == 0){
21       a[bn] = addr = balloc(ip->dev);
22       log_write(bp);
23     }
24     brelse(bp);
25     return addr;
26   }
27   bn -= NINDIRECT;
28
29   if (bn < NDINDIRECT)
30   {
31     if((addr = ip->addrs[NDIRECT + 1]) == 0)
32       ip->addrs[NDIRECT + 1] = addr = balloc(ip->dev);
33     bp = bread(ip->dev, addr);
34     a = (uint*)bp->data;
35     if((addr = a[bn/NINDIRECT]) == 0) {
36       a[bn/NINDIRECT] = addr = balloc(ip->dev);
37       log_write(bp);
38     }
39     brelse(bp);
40     // 重复上面的代码, 实现二级索引
41           bp = bread(ip->dev, addr);
42     a = (uint*)bp->data;
43     if ((addr = a[bn%NINDIRECT]) == 0) {
44       a[bn%NINDIRECT] = addr = balloc(ip->dev);
45       log_write(bp);
```

```
    }
    brelse(bp);
    return addr;
  }

  panic("bmap: out of range");
}
```

(3) **扩充** itrunc() **函数**

在 itrunc() 函数中添加释放二级索引指向的块的代码:

```
void
itrunc(struct inode *ip)
{
  int i, j;
  struct buf *bp, *bp2;
  uint *a, *a2;

  for(i = 0; i < NDIRECT; i++){
    if(ip->addrs[i]){
      bfree(ip->dev, ip->addrs[i]);
      ip->addrs[i] = 0;
    }
  }

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

  // 释放二级索引指向的块
  if(ip->addrs[NDIRECT + 1]){
    bp = bread(ip->dev, ip->addrs[NDIRECT + 1]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT; j++){
```

```
32        if(a[j])
33        {
34          bp2 = bread(ip->dev, a[j]);
35          a2 = (uint*)bp2->data;
36          for(i = 0; i < NINDIRECT; i++) {
37            if(a2[i]) bfree(ip->dev, a2[i]);
38          }
39          brelse(bp2);
40          bfree(ip->dev, a[j]);
41          a[j] = 0;
42        }
43      }
44      brelse(bp);
45      bfree(ip->dev, ip->addrs[NDIRECT]);
46      ip->addrs[NDIRECT] = 0;
47    }
48
49    ip->size = 0;
50    iupdate(ip);
51  }
```

(5) **测试结果**

上述操作完成后运行 bigfile 以及 usertests 可以得到以下结果:

```
$ bigfile
........................................
........................................
........................................
........................................
........................................
wrote 65803 blocks
bigfile done; ok
```

图 2: bigfile passed

```
test exitwait: OK
test rmdot: OK
test fourteen: OK
test bigfile: OK
test dirfile: OK
test iref: OK
test forktest: OK
test bigdir: OK
ALL TESTS PASSED
```

图 3: usertests passed

Part 3: Symbolic links

3.1 问题简述

本部分要求我们在 xv6 中实现软链接。

3.2 知识基础

新增加一种 T_SYMLINK 类型的文件，这个文件中存有需要链接到的文件的 pathname，当使用 open 并指定 O_NOFOLLOW 为 0 时，可以打开这个软链接文件指向的文件，而非软链接文件本身。要求实现一个 symlink(char *target, char *path) 这个 syscall，实现 path 所代表的文件软链接到 target 代表的文件，target 不存在也可以。修改 open，从而可以打开软链接文件。注意：软链接文件指向的文件可能也是一个软链接文件，open 需要递归地找到最终的非软链接文件，但是可能出现软链接文件互相链接的情况，会产生死循环，因此规定递归查找软链接的深度不能超过 10.

3.3 实验步骤

(1) **第一步，增加** symlink **系统调用**

user/usys.pl:

```
entry("symlink");
```

user/user.h

```
int symlink(char*, char*);
```

kernel/syscall.h

```
#define SYS_symlink  22
```

kernel/syscall.c

```
extern uint64 sys_symlink(void);
...
[SYS_symlink]    sys_symlink,
```

(2) **增加标志位**

按照提示，增加标志位：

kernel/stat.h 中添加 T_SIMLINK:

```
#define T_SYMLINK 4    // Symbolic Link
```

kernel/fcntl.h 中添加 O_NOFOLLOW, 该 flag 不能和其他 flag 的位重叠:

```
#define O_NOFOLLOW      0x800
```

(3) **实现** symlink() **函数**

根据提示以及 xv6 book 中 8.14 节的知识，可以写出 symlink() 函数，首先要判断是否存在 path 所代表的 inode，如果不存在就用 create 添加一个 T_SYMLINK 类型的 inode。在 inode 的最后添加需要软链接到的 target 的路径名称:

```
uint64
sys_symlink(void)
{
  char target[MAXPATH], path[MAXPATH];
  struct inode *ip;
  // 读取参数
  if (argstr(0, target, MAXPATH) < 0 || argstr(1, path, MAXPATH) < 0)
  {
    return -1;
  }
  // 开启事务
  begin_op();
  if ((ip = namei(path)) == 0) {
    // path的inode不存在
    ip = create(path, T_SYMLINK, 0, 0);
    iunlock(ip);
  }
  ilock(ip);
  // 在符号链接的 data 中写入被链接的文件
  if (writei(ip, 0, (uint64)target, ip->size, MAXPATH) != MAXPATH) {
    panic("symlink");
  }
  iunlockput(ip);
  end_op();
  return 0;
}
```

(4) **改写** sys_open() **函数**

修改 open，添加对 T_SYMLINK 类型文件的处理方法:

```
  int depth = 0;
...
```

```
3   while(ip->type == T_SYMLINK && !(omode & O_NOFOLLOW)) {
4     // 如果访问深度过大，则退出
5     if (depth++ >= 10) {
6       iunlockput(ip);
7       end_op();
8       return -1;
9     }
10    // 读取对应的 inode
11    if(readi(ip, 0, (uint64)path, 0, MAXPATH) < MAXPATH) {
12      iunlockput(ip);
13      end_op();
14      return -1;
15    }
16    iunlockput(ip);
17    // 根据文件名称找到对应的 inode
18    if((ip = namei(path)) == 0) {
19      end_op();
20      return -1;
21    }
22    ilock(ip);
23  }
```

Part 4: 实验结果

运行 make grade，检验结果：

```
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL T
make[1]: 离开目录“/home/ubuntu/桌面/xv6-labs-2020”
== Test running bigfile ==
$ make qemu-gdb
running bigfile: OK (173.7s)
== Test running symlinktest ==
$ make qemu-gdb
(3.0s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (346.0s)
== Test time ==
time: OK
Score: 100/100
ubuntu@VM5153-OSVM:~/桌面/xv6-labs-2020$
```

图 4: Result

本次实验完成了 File System 有关功能的实现和实际应用。在实验过程中得到了以下收获：

- 对于文件系统有了更深刻的理解
- 了解了文件系统实现的具体方法以及原理，深入体会文件系统的内部结构。
- 认识到了文件系统设计合理性以及正确性