

实验报告

Lab FileSystem

姓名：田鑫

班级：信息安全

学号：19307110353

Part A: Large files

实验结果:

[illegible]

实验步骤:

1、理解 inode 与 dinode，然后修改定义。

本实验要求将 12+256 的大小限制拓展为 11+256+256*256 的大小限制，实际上也就是将原来的一个直接的 `addrs` 改成再扩展两次的 `doubly-indirect`。所以修改如下：

```

27 #define NDIRECT 11 // 12->11
28 #define NINDIRECT (BSIZE / sizeof(uint))
29 #define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT*NINDIRECT) //add
30
31 // On-disk inode structure
32 struct dinode {
33     short type; // File type
34     short major; // Major device number (T_DEVICE only)
35     short minor; // Minor device number (T_DEVICE only)
36     short nlink; // Number of links to inode in file system
37     uint size; // Size of file (bytes)
38     uint addrs[NDIRECT+2]; // Data block addresses
39 };

```

首先是直接映射的数量变为了 11 个，然后是文件的最大大小改变。

Addr 前 11 个存直接映射，第 12 个存 indirect，最后一个存 double-indirect。inode 只需要改 addr 数组的长度即可。

2、理解并修改 bmap()。

bmap 的功能是对指定的 inode 寻找其第 bn 个 block 的位置。所以

原 bmap 分情况讨论，对前 12 个直接用 addr 里的地址即可（如果不存在需要先分配），对后 256 个则是先找到 addr[12]指向的数据，然后再看是哪一块。所以修改后的 bmap 只需要增加对 double-indirect 的判断即可。代码如下：

```
if(bn < NINDIRECT * NINDIRECT) {
    if((addr = ip->addrs[NINDIRECT+1]) == 0) //fenpei diyceng dakua
        ip->addrs[NINDIRECT+1] = addr = balloc(ip->dev);
    //fenpei dierceng
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn/NINDIRECT]) == 0){
        addr = balloc(ip->dev);
        if(addr){
            a[bn/NINDIRECT] = addr;
            log_write(bp);
        }
    }
    brelse(bp);
    //fenpei disanceng
    bn %= NINDIRECT;
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn]) == 0){
        addr = balloc(ip->dev);
        if(addr){
            a[bn] = addr;
            log_write(bp);
        }
    }
    brelse(bp);
    return addr;
}
```

3、修改 itrunc()

仿照对 indirect 的释放添加对 double-indirect 的释放就好。

Part B: Symbolic links

实验结果：

```

$ symlintest
Start: test symlinks
FAILURE: failed to stat b
Start: test concurrent symlinks
test concurrent symlinks: ok

```

实验步骤：

1、添加系统调用，具体代码之后实现：

这步做了很多次了，不过多说明。

2、增加新的文件类型 T_SYMLINK 与新的 flag O_NOFOLLOW。

这里对 O_NOFOLLOW 的值并没有什么要求。

```

kernel > C /usr/include > O_CREATE
1  #define O_RDONLY 0x000
2  #define O_WRONLY 0x001
3  #define O_RDWR 0x002
4  #define O_CREATE 0x200
5  #define O_TRUNC 0x400
6  #define O_NOFOLLOW 0x600

```

```

kernel > C stat.h > T_SYMLINK
1  #define T_DIR 1 // Directory
2  #define T_FILE 2 // File
3  #define T_DEVICE 3 // Device
4  #define T_SYMLINK 4 // symbol link

```

3、实现 symlink 函数：

首先参考其他代码看如何创建一个文件

```

391 uint64
392 sys_mkdir(void) //cankad
393 {
394     char path[MAXPATH];
395     struct inode *ip;
396
397     begin_op();
398     if(argstr(0, path, MAXPATH) < 0 || (ip = create(path, T_DIR, 0,
399         end_op();
400     return -1;
401 }
402 iunlockput(ip);
403 end_op();
404 return 0;
405 }

```

所以仿照上面的代码先创建目标文件。

```

528     struct inode *ip;
529     char target[MAXPATH], path[MAXPATH];
530     if(argstr(0, target, MAXPATH) < 0 || argstr(1, path, MAXPATH)
531         return -1;
532
533     begin_op();
534
535     ip = create(path, T_SYMLINK, 0, 0);
536     if(ip == 0){
537         end_op();
538         return -1;
539     }

```

然后再看如何往其中写入数据，这里参考的是 `filewrite` 函数：

```

163         ilock(f->ip);
164         if ((r = writei(f->ip, 1, addr + i, f->off, nl)) > 0)
165             f->off += r;
166         iunlock(f->ip);

```

调用 `writei` 函数即可

```

0
1     // use first data block to save path
2     if(writei(ip, 0, (uint64)target, 0, strlen(target)) < 0) {
3         end_op();
4         return -1;
5     }

```

最后必须补一句

```

547     iunlockput(ip);

```

不能用 `iunlock`，不然会卡住。

4、修改 `sys_open` 函数：

增加一段，先判断是否是符号链接，如果是符号链接，就递归直到超出阈值或不是符号链接

```

325 } else {
326     int symlink_depth = 0;
327     while(1) {
328         if((ip = namei(path)) == 0){
329             end_op();
330             return -1;
331         }
332         ilock(ip);
333         if(ip->type == T_SYMLINK && (omode & O_NOFOLLOW) == 0) {
334             if(++symlink_depth > 10) {
335                 iunlockput(ip);
336                 end_op();
337                 return -1;
338             }
339             if(readi(ip, 0, (uint64)path, 0, MAXPATH) < 0) {
340                 iunlockput(ip);
341                 end_op();
342                 return -1;
343             }
344             iunlockput(ip);
345         } else {
346             break;
347         }
348     }

```

实验感想：

这次实验的代码量比上个实验要多，但难度并不大。主要的难点都在于理解文件系统如何执行以及找到相应的代码来完成功能。通过这次实验，可以加深对文件系统的理解，对 data block 的理解。