Lab 6

Part A: Large files

首先在 fs.h 中按照题目要求修改相关宏与结构体定义，将 direct inode 数量更改为 11 并加入一个 doubly-indirect inode。同时，添加宏定义 NININDIRECT:

```
#define NININDIRECT (BSIZE / sizeof(uint)) * (BSIZE / sizeof(uint))
```

其表示的是 doubly-indirect inode 所占据的数据块总数，值为 NINDIRECT 的平方。之后，在 fs.c 中修改 bmap 映射部分内容，添加二级 block 映射。模仿一级 block，先对块序号减去 NINDIRECT，随后判断 bn 是否超出二级 block 数，若超出则陷入 panic。定义变量 index、offset，分别对应 bn 所表示的数据块对应的一级 block 序号以及在该 block 下的偏移量。随后访问相应 block 中数据，首先通过 NINDIRECT+1 为下标访问 inode 中的数据内容，随后通过 index 访问一级 block 内容，并最终以 offset 得到数据块对应的地址(若为零则为其新分配一块 block 地址空间)。修改部分相关代码如下。

```
bn -= NINDIRECT;

  if(bn < NININDIRECT){
    int index = bn / NINDIRECT;
    int offset = bn % NINDIRECT;
    if((addr = ip->addrs[NDIRECT + 1]) == 0)
      ip->addrs[NDIRECT + 1] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[index]) == 0){
      a[index] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);

    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[offset]) == 0){
      a[offset] = addr = balloc(ip->dev);
      log_write(bp);
    }
```

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

随后在 itrunc 函数中为 doubly-indirect inode 添加回收部分内容。首先与先前类似，以 NINDIRECT+1 为下标访问 doubly-direct inode ,之后通过一个二层循环遍历该 inode 下对应的所有 block，若地址非零则释放先前分配的地址空间。代码如下。

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

运行结果如下：

```
xv6 kernel is booting

init: starting sh
$ bigfile
..........................................................................................................................
..........................................................................................................................
..........................................................................................................................
..........................................................................................................................
..........................................................................................................................
.....................
wrote 65803 blocks
bigfile done; ok
$
```

Part B: Symbolic links

首先添加系统调用 symlink 相关定义（user/usys.pl 中添加 entry，user/user.h 中添加定义，kernel/syscall.h 中添加宏定义，kernel/syscall.c 中添加掩码及句柄定义），并根据提

示在 fcntl.c 中添加宏定义 O_NOFOLLOW。之后，在 sysfile.c 中添加 sys_symlink 系统调用函数。首先在 create 函数中添加 T_SYMLINK 类型的创建方式（直接返回结构体 ip 即可），之后在 sys_symlink 中可直接调用 create 创建象征链接 inode，之后使用 writei 函数向 ip 中写入被链接的路径即可。

```
uint64
sys_symlink(void)
{
  char target[MAXPATH], path[MAXPATH];
  struct inode *ip;
  if(argstr(0, target, MAXPATH) < 0 || argstr(1, path, MAXPATH) < 0) return -1;

  begin_op();
  ip = create(path, T_SYMLINK, 0, 0);
  if(ip == 0){
    end_op();
    return -1;
  }

  if(writei(ip, 0, (uint64)target, 0, MAXPATH) != MAXPATH) return -1;

  iunlockput(ip);
  end_op();
  return 0;
}
```

随后修改 sys_open 部分内容，在其中添加象征链接的访问方式，进入象征链接访问的条件为当前模式并非O_NOFOLLOW并且ip的类型为SYMLINK。定义变量depth表示symlink深度，target 表示当前象征链接下的目标路径，并通过 namei 函数让 ip 指向下一层，即当前 inode 的象征链接目标路径对应 inode，以此对 ip 循环更新，并注意查看 ip 类型：若某次循环时 ip 类型不再是 symlink，说明访问到真实数据块，因此可以跳出循环。每次循环对 depth 进行判断，若深度大于十则直接返回-1 表示访问失败。

```
if(!(omode & O_NOFOLLOW) && ip->type == T_SYMLINK){
    int depth = 0;
    char target[MAXPATH];
    while(ip->type == T_SYMLINK){
```

```
      if(depth == 10){
        iunlockput(ip);
        end_op();
        return -1;
      }
      depth++;
      memset(target, 0, sizeof(target));
      readi(ip, 0, (uint64)target, 0, MAXPATH);
      iunlockput(ip);
      if((ip = namei(target)) == 0){
        end_op();
        return -1;
      }
      ilock(ip);
    }
  }
```

运行结果如下：

```
xv6 kernel is booting

init: starting sh
$ symlinktest
Start: test symlinks
test symlinks: ok
Start: test concurrent symlinks
test concurrent symlinks: ok
$
```

Thoughts

本次实验实现了二级 indirect inode 以及文件的象征性链接，加深了对 xv6 文件系统的理解。