Lab6: file system

PB20050987 梁兆懿

实验六需要我们阅读xv6课本的第八章，实现添加大文件以及实现文件的符号链接。实验需要跳转到fs分支：

```
  $ git fetch
  $ git checkout fs
  $ make clean
```

Large files

在第一部分的实验中，我们需要增加xv6文件的最大大小。xv6文件原本的限制为268给块，我们需要将其更改，使其可以创建包含 65803 个块的文件。

实验需要用二级块索引实现。

实验代码

由提示知，我们需要把直接块号的宏定义NDIRECT修改为11个。二级间接块号的总数修改为一级间接块号的平方，最大文件数也需要修改为二级索引后的值：

```
//修改fs.h

#define NDIRECT 11
#define NDOUBLYINDIRECT (NINDIRECT * NINDIRECT)
#define MAXFILE (NDIRECT + (NINDIRECT) + (NDOUBLE))
```

接下来修改 inode 相关结构体的块号数组，由于实际inode的块号总数没有变，而NDRECT减少了1。因此需要将struct inode的 addrs 数组大小设置为 NDIRECT+2。

```
//修改fs.h
struct dinode {
  uint addrs[NDIRECT+2];
};

//修改file.h
struct inode {
  uint addrs[NDIRECT+2];
};
```

然后根据提示修改fs.c 的bmap函数，该函数用于返回 inode 的相对块号对应的磁盘中的块号.由于 inode 结构中前 NDIRECT 个块号与修改前是一致的, 因此只需要添加对第 NDIRECT 即 13 个块的二级间接索引的处理代码.。处理的方法与处理第 NDIRECT 个块号即一级间接块号的方法是类似的, 只是需要索引两次，引用第一次处理的代码修改即可：

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

```
  uint addr, *a0, *a1;
  struct buf *bp0, *bp1;

  if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0)
      ip->addrs[bn] = addr = balloc(ip->dev);
    return addr;
  }
  bn -= NDIRECT;

  if(bn < NINDIRECT){
    // Load indirect block, allocating if necessary.
    if((addr = ip->addrs[NDIRECT]) == 0)
      ip->addrs[NDIRECT] = addr = balloc(ip->dev);
    bp0 = bread(ip->dev, addr);
    a0 = (uint*)bp0->data;
    if((addr = a0[bn]) == 0){
      a0[bn] = addr = balloc(ip->dev);
      log_write(bp0);
    }
    brelse(bp0);
    return addr;
  }
  bn -= NINDIRECT;


  if (bn < NDOUBLE) {
    if ((addr = ip->addrs[NDIRECT + 1]) == 0)
      ip->addrs[NDIRECT + 1] = addr = balloc(ip->dev);
    bp0 = bread(ip->dev, addr);
    a0 = (uint*)bp0->data;
    int index = bn / NINDIRECT;
    int remain = bn % NINDIRECT;
    if ((addr = a0[index]) == 0) {
      a0[index] = addr = balloc(ip->dev);
      log_write(bp0);
    }
    brelse(bp0);

    bp1 = bread(ip->dev, addr);
    a1 = (uint*)bp1->data;
    if ((addr = a1[remain]) == 0) {
      a1[remain] = addr = balloc(ip->dev);
      log_write(bp1);
    }

    brelse(bp1);
    return addr;
  }

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

接下来需要修改`itrunc()`函数。因为添加了二级间接块的结构, 所以也需要添加对该部分的块的释放的代码。`itrunc()`函数实现数据块的释放，释放的方式同一级间接块号的结构, 只需要两重循环去分别遍历二级间接块以及其中的一级间接块，参考一级数据块释放代码即可。

```
void
itrunc(struct inode *ip)
{
  int i, j, k;
  struct buf *bp0, *bp1;
  uint *a0, *a1;

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

  if(ip->addrs[NDIRECT]){
    bp0 = bread(ip->dev, ip->addrs[NDIRECT]);
    a0 = (uint*)bp0->data;
    for(j = 0; j < NINDIRECT; j++){
      if(a0[j])
        bfree(ip->dev, a0[j]);
    }
    brelse(bp0);
    bfree(ip->dev, ip->addrs[NDIRECT]);
    ip->addrs[NDIRECT] = 0;
  }


  if (ip->addrs[NDIRECT + 1]) {
    bp0 = bread(ip->dev, ip->addrs[NDIRECT + 1]);
    a0 = (uint*)bp0->data;
    for (j = 0; j < NINDIRECT; j++) {
      if (a0[j]) {
        bp1 = bread(ip->dev, a0[j]);
        a1 = (uint*)bp1->data;
        for (k = 0; k < NINDIRECT; k++) {
          if (a1[k]) {
            bfree(ip->dev, a1[k]);
          }
        }
        brelse(bp1);
        bfree(ip->dev, a0[j]);
      }
    }

    brelse(bp0);
    bfree(ip->dev, ip->addrs[NDIRECT + 1]);
    ip->addrs[NDIRECT + 1] = 0;
  }

  ip->size = 0;
  iupdate(ip);
}
```

实验结果

Symbolic links

在本练习中，需要向 xv6 添加符号链接。符号链接（或软链接）按路径名引用链接的文件。打开符号链接时，内核会跟随指向引用文件的链接。

实验代码

根据提示，我们需要添加有关 `symlink` 系统调用的定义声明。在`kernel/syscall.h`，`kernel/syscall.c`，`kernel/stat.h`， `user/usys.pl` 和 `user/user.h`中添加：

```
//修改kernel/stat.h
...
#define T_SYMLINK 4

//修改kernel/syscall.h
...
#define SYS_symlink 22

//修改user/user.h
...
int symlink(const char*, const char*);

//修改user/usys.pl
...
entry("symlink");

//修改kernel/syscall.c
extern uint64 sys_symlink(void);
...

static uint64 (*syscalls[])(void) = {
...
[SYS_symlink]   sys_symlink, //+++++++++
};
```

然后添加可执行文件到Makefile以及kernel/fcntl.h：

```
//修改MAKEFILE
$U/_symlinktest\

//修改kernel/fcntl.h
#define O_NOFOLLOW   0x800
```

根据提示，我们在`kernel/sysfile.c`中实现`sys_symlink()`函数。先创建符号链接路径对于的inode结构，然后再用writei（）函数将目标文件路径写入inode，再调用`iunlockput()`来释放 inode 的锁和其本身。

```
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;
}
```

最后修改打开链接文件函数sys_open，增加对符号链接文件的处理：

```
uint64
sys_open(void)
{
  char path[MAXPATH];
  int fd, omode;
  struct file *f;
  struct inode *ip;
  int n;

  if((n = argstr(0, path, MAXPATH)) < 0 || argint(1, &omode) < 0)
    return -1;

  begin_op();

  if(omode & O_CREATE){
    ip = create(path, T_FILE, 0, 0);
    if(ip == 0){
      end_op();
      return -1;
    }
```

```
  } else {
    if((ip = namei(path)) == 0){
      end_op();
      return -1;
    }
    ilock(ip);
    if(ip->type == T_DIR && omode != O_RDONLY){
      iunlockput(ip);
      end_op();
      return -1;
    }
  }

  if(ip->type == T_DEVICE && (ip->major < 0 || ip->major >= NDEV)){
    iunlockput(ip);
    end_op();
    return -1;
  }
//增加对符号链接文件的处理
  if (ip->type == T_SYMLINK && !(omode & O_NOFOLLOW)) {
    int cnt = 0;
    while (ip->type == T_SYMLINK) {
      if (readi(ip, 0, (uint64)&path, 0, MAXPATH) == -1) {
        iunlockput(ip);
        end_op();
        return -1;
      }
      iunlockput(ip);
      if ((ip = namei(path)) == 0) {
        end_op();
        return -1;
      }
      cnt++;
      if (cnt == 10) {
        end_op();
        return -1;
      }
      ilock(ip);
    }
  }

  if((f = filealloc()) == 0 || (fd = fdalloc(f)) < 0){
    if(f)
      fileclose(f);
    iunlockput(ip);
    end_op();
    return -1;
  }

  if(ip->type == T_DEVICE){
    f->type = FD_DEVICE;
    f->major = ip->major;
  } else {
    f->type = FD_INODE;
    f->off = 0;
  }
  f->ip = ip;
  f->readable = !(omode & O_WRONLY);
```

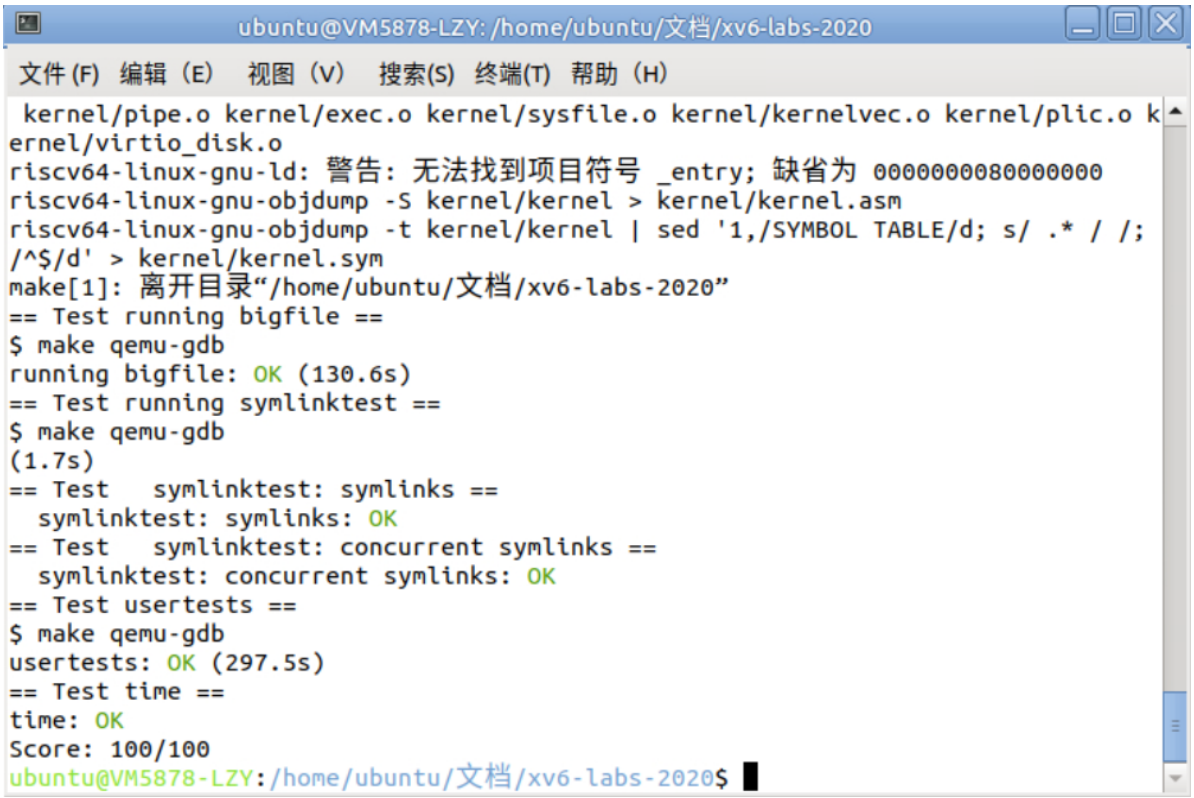
```
  f->writable = (omode & O_WRONLY) || (omode & O_RDWR);

  if((omode & O_TRUNC) && ip->type == T_FILE){
    itrunc(ip);
  }

  iunlock(ip);
  end_op();

  return fd;
}
```

实验结果



实验感受与收获

最后一次实验的内容主要是对文件系统的考察，尤其是对块的间接索引的理解，此外还考察了软连接的理解以及实现，在xv6的参考书上都有详细的介绍。因此本次实验在提示的帮助下较为顺利，需要注意的时在此过程中对锁的调用以及放开，以及不能漏掉`symlink`的各种声明。