# Lab9 file system

### 实验提要

在本实验中，您将向xv6文件系统添加大文件和符号链接。

## Large files

### 实验目的

In this assignment you'll increase the maximum size of an xv6 file. Currently xv6 files are limited to 268 blocks, or 268\*BSIZE bytes (BSIZE is 1024 in xv6). This limit comes from the fact that an xv6 inode contains 12 "direct" block numbers and one "singly-indirect" block number, which refers to a block that holds up to 256 more block numbers, for a total of 12+256=268 blocks.

在此作业中，您将增加xv6文件的最大大小。 当前，xv6文件限制为268个块或268 \* BSIZE字节（xv6中`BSIZE`为1024）。 此限制来自以下事实：xv6索引节点包含12个“直接”块编号和一个“单间接”块编号，这是指最多容纳256个以上块编号的块，总共12 + 256 = 268 块。

简言之就是将xv6原先的12个直接索引和1个一级索引改为11和直接索引+1个一级索引+1个二级索引.

### 实验步骤

1. 将fs.h中NDIRECT从12修改为11，再定义二级页表，修改最大文件

NDINDIRECT表示二级索引块号的总数,能够表示的块号个数是一级索引的平方256\*256。

#define NDIRECT 11

#define NINDIRECT (BSIZE / sizeof(uint))

#define NDINDIRECT (NINDIRECT \* NINDIRECT)

#define MAXFILE (NDIRECT + NINDIRECT + NDINDIRECT)

1. 修改denode和inode，将addrs的大小加1

  uint addrs[NDIRECT+2];

1. 修改bmap

根据指导，现在要添加一个二级索引，那么可以模仿一级索引的实现，在一级索引中再进行一次一级索引的操作。

static uint

bmap(struct inode \*ip, uint bn)

{

  uint addr, \*a;

  struct buf \*bp;

  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);

    bp = bread(ip->dev, addr);

    a = (uint\*)bp->data;

    if((addr = a[bn]) == 0){

      a[bn] = addr = balloc(ip->dev);

      log\_write(bp);

    }

    brelse(bp);

    return addr;

  }

  // 实现二级索引

  bn -= NINDIRECT;

  if(bn < NDOUBLYINDIRECT) {

    // 首先获取锁

    if((addr = ip->addrs[NDIRECT + 1]) == 0) {

      ip->addrs[NDIRECT + 1] = addr = balloc(ip->dev);

    }

    bp = bread(ip->dev, addr);

    a = (uint\*)bp->data;

    // 获取一级索引地址

    if((addr = a[bn / NINDIRECT]) == 0) {

      a[bn / NINDIRECT] = addr = balloc(ip->dev);

      log\_write(bp);

    }

    brelse(bp);

    bp = bread(ip->dev, addr);

    a = (uint\*)bp->data;

    bn %= NINDIRECT;

    // 获取直接索引地址

    if((addr = a[bn]) == 0) {

      a[bn] = addr = balloc(ip->dev);

      log\_write(bp);

    }

    brelse(bp);

    return addr;

  }

  panic("bmap: out of range");

}

1. 修改itrunc，增加用于释放二级页表数据块的功能

void

itrunc(struct inode \*ip)

{

  int i, j;

  struct buf \*bp;

  uint \*a;

  //new code for 二级页表

  int k;

  struct \*bp2;

  uint \*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;

  }

  // new code for 二级页表

  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]) {

        bp2 = bread(ip->dev, a[j]);

        a2 = (uint\*)bp2->data;

        for(k = 0; k < NINDIRECT; ++k) {

          if(a2[k]) {

            bfree(ip->dev, a2[k]);

          }

        }

        brelse(bp2);

        bfree(ip->dev, a[j]);

        a[j] = 0;

      }

    }

    brelse(bp);

    bfree(ip->dev, ip->addrs[NDIRECT + 1]);

    ip->addrs[NDIRECT + 1] = 0;

  }

  ip->size = 0;

  iupdate(ip);

}

1. 在终端输入make qemu启动xv6后键入指令bigfile，得到的结果符合预期

### 实验中遇到的困难和解决办法

因为不熟练，头文件又忘记添加了。

不知道该怎么实现二级页表：模仿一级页表的实现。

### 实验小结

本身这个小实验在概念上不难理解，在实现上只要会模仿一级页表的实现逻辑来实现二级页表就好了。就算有一些变量不明白具体作用也可以完成该实验，但完成了整个实验后还是好好细致地看课查资料理清了实现一级页表时每一步在做什么。

## Symbolic links

### 实验目的

In this assignment you'll increase the maximum size of an xv6 file. Currently xv6 files are limited to 268 blocks, or 268\*BSIZE bytes (BSIZE is 1024 in xv6). This limit comes from the fact that an xv6 inode contains 12 "direct" block numbers and one "singly-indirect" block number, which refers to a block that holds up to 256 more block numbers, for a total of 12+256=268 blocks.

在本练习中，您将向xv6添加符号链接。 符号链接（或软链接）通过路径名引用链接的文件。 当打开符号链接时，内核将链接指向引用的文件。 符号链接类似于硬链接，但是硬链接仅限于指向同一磁盘上的文件，而符号链接可以跨磁盘设备。 尽管xv6不支持多个设备，但是实现此系统调用是了解路径名查找如何工作的一个好习惯。

### 实验步骤

1. 仿照前面的lab，添加一个sysylink系统调用

包括在syscall.h中添加

#define SYS\_symlink 22

在syscall.c中添加

extern uint64 sys\_symlink(void);

[SYS\_symlink] sys\_symlink,

在usys.pl中添加

entry("symlink");

在user.h中添加

int symlink(char \*target, char \*path);

1. 在stat.h中新增文件类型’T\_SYMLINK’

#define T\_SYMLINK 4

1. 在fcntl.h中新增文件标志位’O\_NOFOLLOW’

#define O\_NOFOLLOW   0x600

1. 实现sys\_symlink()

该函数实现了一个用于创建符号链接的系统调用首先，函数会检查从用户空间获取的目标路径和链接路径是否有效，如果无效则返回 -1 表示出错。接着，函数调用 begin\_op() 函数，这可能是一个开始文件系统操作的函数，用于确保操作的原子性和一致性。然后，函数尝试创建一个符号链接。它调用了 create 函数，并传递了链接路径 path、文件类型 T\_SYMLINK（可能是一个常量，表示符号链接类型），以及额外的两个参数设为 0。如果创建成功，则 create 函数会返回一个指向新创建的 inode 的指针，将其赋值给 ip 变量。接下来，函数将目标路径 target 的内容写入新创建的 inode 中。如果写入的字节数与目标路径的长度不匹配，则表示写入出错，函数会释放已分配的 inode 并返回 -1 表示出错。如果写入成功，则释放 inode 并结束文件系统操作，最后返回 0 表示符号链接创建成功。

uint64

sys\_symlink(void) {

  char target[MAXPATH], path[MAXPATH];

  struct inode \*ip;

  int n;

  if ((n = argstr(0, target, MAXPATH)) < 0

    || argstr(1, path, MAXPATH) < 0) {

    return -1;

  }

  begin\_op();

  if((ip = create(path, T\_SYMLINK, 0, 0)) == 0) {

    end\_op();

    return -1;

  }

  if(writei(ip, 0, (uint64)target, 0, n) != n) {

    iunlockput(ip);

    end\_op();

    return -1;

  }

  iunlockput(ip);

  end\_op();

  return 0;

}

1. 修改sys\_open函数

在创建符号链接时，如果目标路径是一个符号链接本身，我们需要递归地获取符号链接的目标路径，直到找到一个非符号链接的路径为止。当搜索次数达到一定次数后表示文件打开失败

  else {

    int max\_depth = 10, depth = 0;

    while (1) {

      if ((ip = namei(path)) == 0) {

        end\_op();

        return -1;

      }

      ilock(ip);

      if (ip->type == T\_SYMLINK && (omode & O\_NOFOLLOW) == 0) {

        if (++depth > max\_depth) {

          iunlockput(ip);

          end\_op();

          return -1;

        }

        if (readi(ip, 0, (uint64)path, 0, MAXPATH) < MAXPATH) {

          iunlockput(ip);

          end\_op();

          return -1;

        }

        iunlockput(ip);

      }

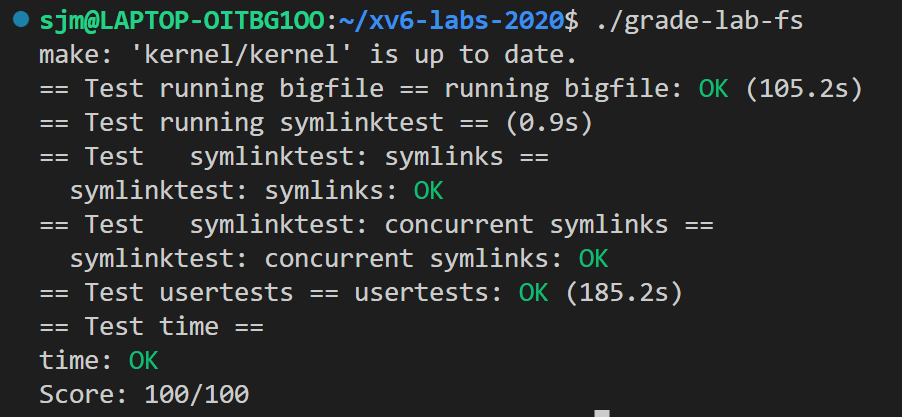
      else

        break;

    }

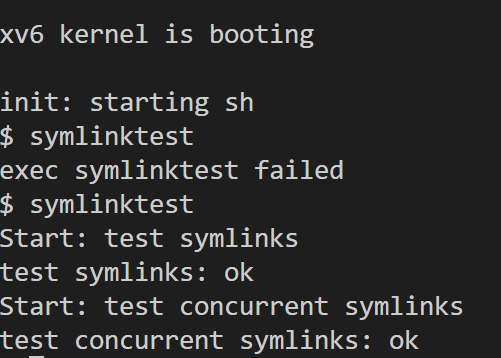
  }

退出xv6，并在终端输入./grade-lab-fs，得到的测试结果符合预期：



### 实验中遇到的问题和解决办法

起初没有在makefile中加入symlinktest指令，导致没办法进行测试，但是在加入makefile后启动make qemu，两次相同输入下却得到了不同的结果，原因可能是xv6运行或退出有一定的时间延迟，需要等待前一次xv6完全退出后才能进行下一次调试。



### 实验心得

文件系统当时在课内考试的时候认真复习了一段时间，可以说做起这个实验来还是很有帮助的，但是这次也从实际感受到了文件系统在代码上具体是怎么实现，加深了一遍印象。但其实课上讲的关于文件系统的知识都还比较浅显，具体工程上的操作系统中文件系统的实现更加复杂，对于本人来说还是任重而道远。