Lab9 file system

1. Large files(moderate)

1）实验目的

> Modify `bmap()` so that it implements a doubly-indirect block, in addition to direct blocks and a singly-indirect block. You'll have to have only 11 direct blocks, rather than 12, to make room for your new doubly-indirect block; you're not allowed to change the size of an on-disk inode. The first 11 elements of `ip->addrs[]` should be direct blocks; the 12th should be a singly-indirect block (just like the current one); the 13th should be your new doubly-indirect block.
> 修改 `bmap()`，使其除了实现直接块和单间接块外，还实现双间接块。你必须只有11个直接区块，而不是12个，才能为你的新的双重间接区块腾出空间；不允许更改磁盘上inode的大小。`ip->addrs[]` 的前11个元素应该是直接块；第12个应该是一个单独的间接块（就像当前的块一

> 样）；13号应该是你新的双间接挡。
> 增加二级索引
> 实现软链接

2）实验步骤

编写代码

xv6系统的inode原先有12个直接索引，1个一级索引，我们将一个直接做因修改为二级索引，即11个直接索引、1个一级索引和1个二级索引。总共可以指向 `11 + 256 + 256*256 = 65803` 个物理块

修改 `kernel/fs.h` 文件中的直接索引块数为11

```
// #define NDIRECT 12
// lab9-1
#define NDIRECT 11
```

NDIRECT变少了但其实总量不变，仍是13，因此需要修改 `kernel/fs.h` 文件中的 `struct dinode` 的 `addrs` 数据成员

```
struct dinode {
  ...
  uint addrs[NDIRECT+2];   // Data block addresses
};
```

同时也要修改 `kelnel/file.h` 文件中的 `struct inode` 的 `addrs` 数据成员

```
struct inode {
  ...
  uint size;
  // lab9-1
  uint addrs[NDIRECT+2];
};
```

在 `kernel/fs.h` 文件中定义二级索引总数，并修改文件最大体积

```
// 二级索引能找到的物理块数量lab9-1
#define NSECDIRECT (NINDIRECT * NINDIRECT)
#define MAXFILE (NDIRECT + NINDIRECT + NSECDIRECT)
```

参考 `kernel/fs.c` 的 `bmap()` 函数进一步添加对于二级索引的处理

```c
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;
  }

  // 到二级索引 lab9-1
  bn -= NINDIRECT;
  if (bn < NSECDIRECT) {
    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[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;
    if ((addr = a[bn]) == 0) {
      a[bn] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    return addr;
    }

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

参考 `kernel/fs.c` 的 `itrunc()` 函数进一步添加对于二级索引的处理

```
void
itrunc(struct inode *ip)
{
  // k, bp1, a1用于二级索引 lab9-1
  int i, j, k;
  struct buf *bp, *bp1;
  uint *a, *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]){
    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;
  }
  // 释放二级索引的物理块
  if (ip->addrs[NDIRECT + 1]) {
    bp = bread(ip->dev, ip->addrs[NDIRECT + 1]);
    a = (uint*)bp->data;
    // 遍历二级索引
    for (j = 0; j < NINDIRECT; ++j) {
      if (a[j]) {
        bp1 = bread(ip->dev, a[j]);
        a1 = (uint*)bp1->data;
        for (k = 0; k < NINDIRECT; ++k) {
          if (a[k]) {
            bfree(ip->dev, a1[k]);
          }
        }
        brelse(bp1);
        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);
  }
```

测试程序

在存放 `Makefile` 文件的目录下执行如下命令

```
$ make qemu
```

启动xv6系统 QEMU模拟器
键入指令 `bigfile` 进行测试

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

测试通过

键入 `usertests` 命令进行测试

```
$ usertests
usertests starting
test manywrites: OK
test execout: OK
test copyin: OK
test copyout: OK
test copyinstr1: OK
test copyinstr2: OK
test copyinstr3: OK
test rwsbrk: OK
test truncate1: OK
test truncate2: OK
test truncate3: OK
test reparent2: OK
test pgbug: OK
test sbrkbugs: usertrap(): unexpected scause 0x000000000000000c pid=3241
            sepc=0x00000000000056a6 stval=0x00000000000056a6
usertrap(): unexpected scause 0x000000000000000c pid=3242
            sepc=0x00000000000056a6 stval=0x00000000000056a6
OK
test badarg: OK
test reparent: OK
test twochildren: OK
test forkfork: OK
test forkforkfork: OK
test argptest: OK
test createdelete: OK
test linkunlink: OK
test linktest: OK
test unlinkread: OK
test concreate: OK
test subdir: OK
test fourfiles: OK
test sharedfd: OK
test dirtest: OK
test exectest: OK
test bigargtest: OK
test bigwrite: OK
test bsstest: OK
test sbrkbasic: OK
test sbrkmuch: OK
test kernmem: usertrap(): unexpected scause 0x000000000000000d pid=6168
            sepc=0x000000000000215e stval=0x0000000080000000
usertrap(): unexpected scause 0x000000000000000d pid=6169
            sepc=0x000000000000215e stval=0x000000008000c350
usertrap(): unexpected scause 0x000000000000000d pid=6170
            sepc=0x000000000000215e stval=0x00000000800186a0
usertrap(): unexpected scause 0x000000000000000d pid=6171
            sepc=0x000000000000215e stval=0x00000000800249f0
usertrap(): unexpected scause 0x000000000000000d pid=6172
```

```
            sepc=0x000000000000215e stval=0x0000000080030d40
usertrap(): unexpected scause 0x000000000000000d pid=6173
            sepc=0x000000000000215e stval=0x000000008003d090
usertrap(): unexpected scause 0x000000000000000d pid=6174
            sepc=0x000000000000215e stval=0x00000000800493e0
usertrap(): unexpected scause 0x000000000000000d pid=6175
            sepc=0x000000000000215e stval=0x0000000080055730
usertrap(): unexpected scause 0x000000000000000d pid=6176
            sepc=0x000000000000215e stval=0x0000000080061a80
usertrap(): unexpected scause 0x000000000000000d pid=6177
            sepc=0x000000000000215e stval=0x000000008006ddd0
usertrap(): unexpected scause 0x000000000000000d pid=6178
            sepc=0x000000000000215e stval=0x000000008007a120
usertrap(): unexpected scause 0x000000000000000d pid=6179
            sepc=0x000000000000215e stval=0x0000000080086470
usertrap(): unexpected scause 0x000000000000000d pid=6180
            sepc=0x000000000000215e stval=0x00000000800927c0
usertrap(): unexpected scause 0x000000000000000d pid=6181
            sepc=0x000000000000215e stval=0x000000008009eb10
usertrap(): unexpected scause 0x000000000000000d pid=6182
            sepc=0x000000000000215e stval=0x00000000800aae60
usertrap(): unexpected scause 0x000000000000000d pid=6183
            sepc=0x000000000000215e stval=0x00000000800b71b0
usertrap(): unexpected scause 0x000000000000000d pid=6184
            sepc=0x000000000000215e stval=0x00000000800c3500
usertrap(): unexpected scause 0x000000000000000d pid=6185
            sepc=0x000000000000215e stval=0x00000000800cf850
usertrap(): unexpected scause 0x000000000000000d pid=6186
            sepc=0x000000000000215e stval=0x00000000800dbba0
usertrap(): unexpected scause 0x000000000000000d pid=6187
            sepc=0x000000000000215e stval=0x00000000800e7ef0
usertrap(): unexpected scause 0x000000000000000d pid=6188
            sepc=0x000000000000215e stval=0x00000000800f4240
usertrap(): unexpected scause 0x000000000000000d pid=6189
            sepc=0x000000000000215e stval=0x0000000080100590
usertrap(): unexpected scause 0x000000000000000d pid=6190
            sepc=0x000000000000215e stval=0x000000008010c8e0
usertrap(): unexpected scause 0x000000000000000d pid=6191
            sepc=0x000000000000215e stval=0x0000000080118c30
usertrap(): unexpected scause 0x000000000000000d pid=6192
            sepc=0x000000000000215e stval=0x0000000080124f80
usertrap(): unexpected scause 0x000000000000000d pid=6193
            sepc=0x000000000000215e stval=0x00000000801312d0
usertrap(): unexpected scause 0x000000000000000d pid=6194
            sepc=0x000000000000215e stval=0x000000008013d620
usertrap(): unexpected scause 0x000000000000000d pid=6195
            sepc=0x000000000000215e stval=0x0000000080149970
usertrap(): unexpected scause 0x000000000000000d pid=6196
            sepc=0x000000000000215e stval=0x0000000080155cc0
usertrap(): unexpected scause 0x000000000000000d pid=6197
            sepc=0x000000000000215e stval=0x0000000080162010
```

```
usertrap(): unexpected scause 0x000000000000000d pid=6198
            sepc=0x000000000000215e stval=0x000000008016e360
usertrap(): unexpected scause 0x000000000000000d pid=6199
            sepc=0x000000000000215e stval=0x000000008017a6b0
usertrap(): unexpected scause 0x000000000000000d pid=6200
            sepc=0x000000000000215e stval=0x0000000080186a00
usertrap(): unexpected scause 0x000000000000000d pid=6201
            sepc=0x000000000000215e stval=0x0000000080192d50
usertrap(): unexpected scause 0x000000000000000d pid=6202
            sepc=0x000000000000215e stval=0x000000008019f0a0
usertrap(): unexpected scause 0x000000000000000d pid=6203
            sepc=0x000000000000215e stval=0x00000000801ab3f0
usertrap(): unexpected scause 0x000000000000000d pid=6204
            sepc=0x000000000000215e stval=0x00000000801b7740
usertrap(): unexpected scause 0x000000000000000d pid=6205
            sepc=0x000000000000215e stval=0x00000000801c3a90
usertrap(): unexpected scause 0x000000000000000d pid=6206
            sepc=0x000000000000215e stval=0x00000000801cfde0
usertrap(): unexpected scause 0x000000000000000d pid=6207
            sepc=0x000000000000215e stval=0x00000000801dc130
OK
test sbrkfail: usertrap(): unexpected scause 0x000000000000000d pid=6215
            sepc=0x00000000000041fe stval=0x0000000000012000
OK
test sbrkarg: OK
test validatetest: OK
test stacktest: usertrap(): unexpected scause 0x000000000000000d pid=6219
            sepc=0x00000000000022ce stval=0x000000000000fb90
OK
test opentest: OK
test writetest: OK
test writebig: OK
test createtest: OK
test openiput: OK
test exitiput: OK
test iput: OK
test mem: OK
test pipe1: OK
test preempt: kill... wait... OK
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
```

键入 `Ctrl+a`，松开，然后键入 `x`，退出xv6系统
退出xv6进行单元测试

```
root@LAPTOP-UER420HO:~/xv6-labs-2020# ./grade-lab-fs  bigfile
make: 'kernel/kernel' is up to date.
== Test running bigfile == running bigfile: OK (35.0s)
```

3）实验中遇到的问题和解决方法

本实验没什么问题，跟着提示走，查询网上资料就好

4）实验心得

这里的实验主要是要求我们实现文件系统的二级索引，非常贴合我们操作系统理论课上所教的内存

对于 `bmap()` 函数，它的功能主要是返回 `inode` 对应的逻辑块号相应的磁盘中的物理块号。首先检查 `bn` 位于索引的哪部分，是直接索引还是一级索引还是二级索引，从而执行相应的操作。这里对于二级索引的操作，我们完全可以参考一级索引的操作进行

主要逻辑为：函数首先检查逻辑块号是否小于 `NDIRECT` （直接索引的数量）。如果是，那就对应直接索引，它会检查 `inode` 中对应位置的地址是否为0，如果是，则调用 `balloc()` 函数分配一个新的物理块，并将其地址存储在 `inode` 中。无论是新分配的还是之前已存在的物理块，最后都会返回该地址。

如果逻辑块号大于等于 `NDIRECT`，但小于 `NINDIRECT` （一级间接索引的数量），则进入第二个if语句块，判断为一级索引。在这里，函数首先加载一级间接块（indirect block），如果该块的地址为0，则调用 `balloc()` 函数分配一个新的物理块，并将其地址存储在 `inode` 中。然后，函数从间接块中读取一个指向物理块的地址数组，并根据逻辑块号找到对应的地址。如果找到的地址为0，则再次调用 `balloc` 函数分配一个新的物理块，并将其地址存储在数组中。最后，函数释放间接块的缓冲区，并返回找到的物理块地址。

如果逻辑块号大于等于 `NINDIRECT`，但小于 `NSECDIRECT` （二级间接索引的数量），则进入第三个if语句块，说明为二级索引。在这里，函数首先加载二级间接块（second-level indirect block），如果该块的地址为0，则调用 `balloc()` 函数分配一个新的物理块，并将其地址存储在 `inode` 中。然后，函数从二级间接块中读取一个指向一级间接块的地址数组，并根据逻辑块号找到对应的一级间接块地址。如果找到的地址为0，则再次调用 `balloc()` 函数分配一个新的物理块，并将其地址存储在一级间接块数组中。接下来，函数释放二级间接块的缓冲区，并加载一级间接块。然后，函数从一级间接块中读取一个指向物理块的地址数组，并根据逻辑块号找到对应的地址。如果找到的地址为0，则再次调用 `balloc()` 函数分配一个新的物理块，并将其地址存储在数组中。最后，函数释放一级间接块的缓冲区，并返回找到的物理块地址

而对于 `itrunc()` 函数，它的功能是释放所给文件的数据块
因为只有两级索引，因此我们这里直接通过循环来逐层释放
利用 `brelse()` 释放临时读取数据的缓冲区；利用 `bfree()` 释放数据块

2. Symbolic links (moderate)

1）实验目的

> You will implement the `symlink(char *target, char *path)` system call, which creates a new symbolic link at path that refers to file named by target. For further information, see the man page symlink. To test, add symlinktest to the Makefile and run it. Your solution is complete when the tests produce the following output (including usertests succeeding).
> In this exercise you will add symbolic links to xv6. Symbolic links (or soft links) refer to a linked file by pathname; when a symbolic link is opened, the kernel follows the link to the referred file. Symbolic links resembles hard links, but hard links are restricted to pointing to file on the same disk, while symbolic links can cross disk devices. Although xv6 doesn't support multiple devices, implementing this system call is a good exercise to understand how pathname lookup works.
> 您将实现 `symlink(char*target, char*path)` 系统调用，该调用在引用由target命名的文件的path处创建一个新的符号链接。有关更多信息，请参阅手册页符号链接。要进行测试，请将symlinktest添加到Makefile并运行它。当测试产生以下输出（包括成功的用户测试）时，您的解决方案就完成了。
> 在本练习中，您将向xv6添加符号链接。符号链接（或软链接）是指按路径名链接的文件；当打开一个符号链接时，内核会跟随该链接指向引用的文件。符号链接类似于硬链接，但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，但实现此系统调用是了解路径名查找如何工作的一个很好的练习。

2）实验步骤

编写代码

添加 `symlink` 系统调用
在 `user/user.h` 文件中声明需要添加的系统调用 `symlink`

```
// lab9-2
int symlink(char* target, char* path);
```

在 `user/usys.pl` 文件中添加上述系统调用的入口

```
# lab9-2
...
entry("uptime");
entry("symlink");
```

在 `kernel/syscall.h` 添加定义

```
// lab9-2
#define SYS_close 21
#define SYS_symlink 22
```

在 `kernel/syscall.c` 添加函数声明

```
// lab9-2
extern uint64 sys_uptime(void);
extern uint64 sys_symlink(void);
...
static uint64 (*syscalls[])(void) = {
...
// lab9-2
[SYS_close]  sys_close,
[SYS_symlink] sys_symlink,
};
```

为了实现软链接，我们需要声明新的一种文件类型
在 `kernel/stat.h` 文件中添加

```
...
#define T_DEVICE  3   // Device
// 软链接文件类型 lab9-2
#define T_SYMLINK 4
...
```

在 `kernel/fcntl.h` 文件中添加新的文件标志位 `O_NOFOLLOW`

```
#define O_NOFOLLOW 0x004
```

实现 `sys_symlink()` 函数，用于生成符号链接，新建一个 `inode` 写入相关数据
在 `kernel/sysfile.c` 中 实现 sys_symlink() 函数

```c
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();
  // 创建符号链接
  if ((ip = create(path, T_SYMLINK, 0, 0) == 0)) {
    end_op();
    return -1;
  }
  // 写入目标路径
  if (writei(ip, 0, (uint64)target, 0, MAXPATH) < MAXPATH) {
    iunlockput(ip);
    end_op();
    return -1;
  }
  iunlockput(ip);
  end_op();
  return 0;
}
```

修改 `kernel/sysfile.c` 文件中的 `sys_open()` 函数

```c
uint64
sys_open(void)
{
  ...
  if(omode & O_CREATE){
    ip = create(path, T_FILE, 0, 0);
    if(ip == 0){
      end_op();
      return -1;
    }
  } else {
    // lab9-2
    const int max_depth = 20;
    int depth = 0;
    while (1) {
      if((ip = namei(path)) == 0){
        end_op();
        return -1;
      }
      ilock(ip);
      if (ip->type == T_SYMLINK && (omode & O_NOFOLLOW) == 0) {
        // 超出最深层数
        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;
      }
    }
  }

  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;
    }
    ...
}
```

测试程序

在存放 `Makefile` 文件的目录下执行如下命令

```
$ make qemu
```

启动xv6系统 QEMU模拟器
键入指令 `symlinktest` 进行测试

```
$ symlinktest
Start: test symlinks
test symlinks: ok
Start: test concurrent symlinks
test concurrent symlinks: ok
```

测试通过
键入指令 `usertests` 进行测试

```
$ usertests
usertests starting
test manywrites: OK
test execout: OK
test copyin: OK
test copyout: OK
test copyinstr1: OK
test copyinstr2: OK
test copyinstr3: OK
test rwsbrk: OK
test truncate1: OK
test truncate2: OK
test truncate3: OK
test reparent2: OK
test pgbug: OK
test sbrkbugs: usertrap(): unexpected scause 0x000000000000000c pid=3243
            sepc=0x00000000000056a6 stval=0x00000000000056a6
usertrap(): unexpected scause 0x000000000000000c pid=3244
            sepc=0x00000000000056a6 stval=0x00000000000056a6
OK
test badarg: OK
test reparent: OK
test twochildren: OK
test forkfork: OK
test forkforkfork: OK
test argptest: OK
test createdelete: OK
test linkunlink: OK
test linktest: OK
test unlinkread: OK
test concreate: OK
test subdir: OK
test fourfiles: OK
test sharedfd: OK
test dirtest: OK
test exectest: OK
test bigargtest: OK
test bigwrite: OK
test bsstest: OK
test sbrkbasic: OK
test sbrkmuch: OK
test kernmem: usertrap(): unexpected scause 0x000000000000000d pid=6170
            sepc=0x000000000000215e stval=0x0000000080000000
usertrap(): unexpected scause 0x000000000000000d pid=6171
            sepc=0x000000000000215e stval=0x000000008000c350
usertrap(): unexpected scause 0x000000000000000d pid=6172
            sepc=0x000000000000215e stval=0x00000000800186a0
usertrap(): unexpected scause 0x000000000000000d pid=6173
            sepc=0x000000000000215e stval=0x00000000800249f0
usertrap(): unexpected scause 0x000000000000000d pid=6174
```

```
               sepc=0x000000000000215e stval=0x0000000080030d40
usertrap(): unexpected scause 0x000000000000000d pid=6175
               sepc=0x000000000000215e stval=0x000000008003d090
usertrap(): unexpected scause 0x000000000000000d pid=6176
               sepc=0x000000000000215e stval=0x00000000800493e0
usertrap(): unexpected scause 0x000000000000000d pid=6177
               sepc=0x000000000000215e stval=0x0000000080055730
usertrap(): unexpected scause 0x000000000000000d pid=6178
               sepc=0x000000000000215e stval=0x0000000080061a80
usertrap(): unexpected scause 0x000000000000000d pid=6179
               sepc=0x000000000000215e stval=0x000000008006ddd0
usertrap(): unexpected scause 0x000000000000000d pid=6180
               sepc=0x000000000000215e stval=0x000000008007a120
usertrap(): unexpected scause 0x000000000000000d pid=6181
               sepc=0x000000000000215e stval=0x0000000080086470
usertrap(): unexpected scause 0x000000000000000d pid=6182
               sepc=0x000000000000215e stval=0x00000000800927c0
usertrap(): unexpected scause 0x000000000000000d pid=6183
               sepc=0x000000000000215e stval=0x000000008009eb10
usertrap(): unexpected scause 0x000000000000000d pid=6184
               sepc=0x000000000000215e stval=0x00000000800aae60
usertrap(): unexpected scause 0x000000000000000d pid=6185
               sepc=0x000000000000215e stval=0x00000000800b71b0
usertrap(): unexpected scause 0x000000000000000d pid=6186
               sepc=0x000000000000215e stval=0x00000000800c3500
usertrap(): unexpected scause 0x000000000000000d pid=6187
               sepc=0x000000000000215e stval=0x00000000800cf850
usertrap(): unexpected scause 0x000000000000000d pid=6188
               sepc=0x000000000000215e stval=0x00000000800dbba0
usertrap(): unexpected scause 0x000000000000000d pid=6189
               sepc=0x000000000000215e stval=0x00000000800e7ef0
usertrap(): unexpected scause 0x000000000000000d pid=6190
               sepc=0x000000000000215e stval=0x00000000800f4240
usertrap(): unexpected scause 0x000000000000000d pid=6191
               sepc=0x000000000000215e stval=0x0000000080100590
usertrap(): unexpected scause 0x000000000000000d pid=6192
               sepc=0x000000000000215e stval=0x000000008010c8e0
usertrap(): unexpected scause 0x000000000000000d pid=6193
               sepc=0x000000000000215e stval=0x0000000080118c30
usertrap(): unexpected scause 0x000000000000000d pid=6194
               sepc=0x000000000000215e stval=0x0000000080124f80
usertrap(): unexpected scause 0x000000000000000d pid=6195
               sepc=0x000000000000215e stval=0x00000000801312d0
usertrap(): unexpected scause 0x000000000000000d pid=6196
               sepc=0x000000000000215e stval=0x000000008013d620
usertrap(): unexpected scause 0x000000000000000d pid=6197
               sepc=0x000000000000215e stval=0x0000000080149970
usertrap(): unexpected scause 0x000000000000000d pid=6198
               sepc=0x000000000000215e stval=0x0000000080155cc0
usertrap(): unexpected scause 0x000000000000000d pid=6199
               sepc=0x000000000000215e stval=0x0000000080162010
```

```
usertrap(): unexpected scause 0x000000000000000d pid=6200
            sepc=0x000000000000215e stval=0x000000008016e360
usertrap(): unexpected scause 0x000000000000000d pid=6201
            sepc=0x000000000000215e stval=0x000000008017a6b0
usertrap(): unexpected scause 0x000000000000000d pid=6202
            sepc=0x000000000000215e stval=0x0000000080186a00
usertrap(): unexpected scause 0x000000000000000d pid=6203
            sepc=0x000000000000215e stval=0x0000000080192d50
usertrap(): unexpected scause 0x000000000000000d pid=6204
            sepc=0x000000000000215e stval=0x000000008019f0a0
usertrap(): unexpected scause 0x000000000000000d pid=6205
            sepc=0x000000000000215e stval=0x00000000801ab3f0
usertrap(): unexpected scause 0x000000000000000d pid=6206
            sepc=0x000000000000215e stval=0x00000000801b7740
usertrap(): unexpected scause 0x000000000000000d pid=6207
            sepc=0x000000000000215e stval=0x00000000801c3a90
usertrap(): unexpected scause 0x000000000000000d pid=6208
            sepc=0x000000000000215e stval=0x00000000801cfde0
usertrap(): unexpected scause 0x000000000000000d pid=6209
            sepc=0x000000000000215e stval=0x00000000801dc130
OK
test sbrkfail: usertrap(): unexpected scause 0x000000000000000d pid=6217
            sepc=0x00000000000041fe stval=0x0000000000012000
OK
test sbrkarg: OK
test validatetest: OK
test stacktest: usertrap(): unexpected scause 0x000000000000000d pid=6221
            sepc=0x00000000000022ce stval=0x000000000000fb90
OK
test opentest: OK
test writetest: OK
test writebig: OK
test createtest: OK
test openiput: OK
test exitiput: OK
test iput: OK
test mem: OK
test pipe1: OK
test preempt: kill... wait... OK
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
```

测试通过

键入 `Ctrl+a` ，松开，然后键入 `x` ，退出xv6系统
退出xv6进行单元测试

```
root@LAPTOP-UER420HO:~/xv6-labs-2020# ./grade-lab-fs
== Test running bigfile == running bigfile: OK (63.2s)
== Test running symlinktest == (0.9s)
== Test   symlinktest: symlinks ==
  symlinktest: symlinks: OK
== Test   symlinktest: concurrent symlinks ==
  symlinktest: concurrent symlinks: OK
== Test usertests == usertests: OK (135.1s)
== Test time ==
time: OK
Score: 100/100
```

3）实验中遇到的问题和解决方法

修改 `sys_open()` 函数错误导致一直无法启动xv6系统，因此我们多找几个网上的参考
`sys_open()` 函数中原来的 `ilock(ip)` 忘记删除，导致多次 `ilock()` 函数的调用，删掉就好了

4）实验心得

这里软链接的实现利用了xv6系统中的 `begin_op()` 和 `end_op()` 函数，他们可以实现文件系统的原子操作，主要用于确保在对文件系统进行修改的时候，不会发生竞争条件或数据不一致的情况，从而保证了原子性和一致性。有点类似数据库系统中的事务的处理方式
而对于 `sys_open()` 函数，其中对于软链接的处理方法如下
该函数主要用于根据给定的路径字符串 `path` 找到对应的文件节点。它通过循环和条件判断来处理可能存在的符号链接，并限制了最大打开的深。首先，我们定义了最大深度 `max_depth` ，用于打开的层数，从而避免过度消耗系统的资源。然后，进入一个无限循环，直到找到目标文件节点或发生错误才会退出循环。在循环中，首先调用 `namei(path)` 函数来查找路径对应的文件节点。如果未找到文件节点，则执行清理操作并返回-1表示失败。如果找到了文件节点，代码会对该节点进行加锁，以确保在访问期间不会被其他进程修改。然后，检查该节点的类型是否为符号链接，并检查打开模式是否允许跟随符号链接。如果需要处理符号链接，代码会检查当前深度是否超过了最大深度 `max_depth` 。如果超过最大深度，则执行清理操作并返回-1表示失败。如果深度未超过最大深度，代码会读取符号链接的内容，并将结果存储在 `path` 变量中。如果读取的内容长度小于 `MAXPATH` ，表示读取失败，同样执行清理操作并返回-1表示失败。最后，代码会解锁并释放文件节点。如果文件节点不是符号链接或者不需要处理符号链接，则跳出循环。